# Programming Engineering

Course 3 – 24 February

adiftene@info.uaic.ro

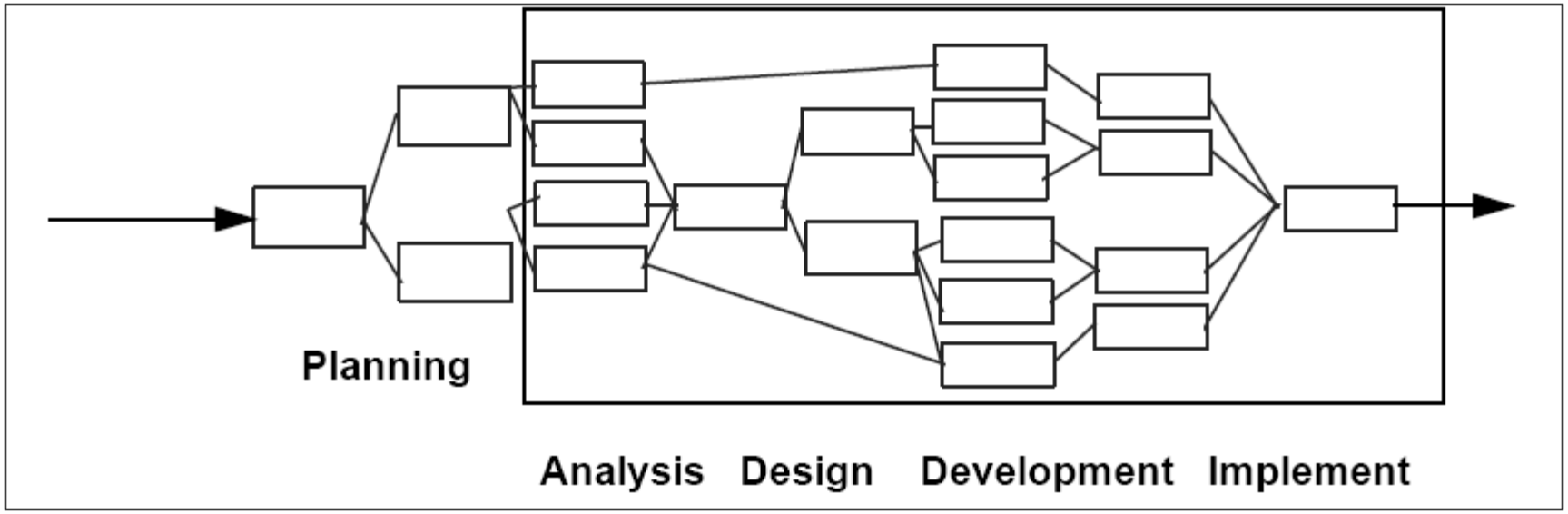
# Content

- From Courses 1, 2…
- Modeling
- Modeling Languages
  - Graphic Languages
- UML – History
- UML – Definition
- UML – Diagram Types
- UML – Use Case Diagram
  - Actors
  - Use Case
- UML – Class Diagrams

# From Courses 1, 2

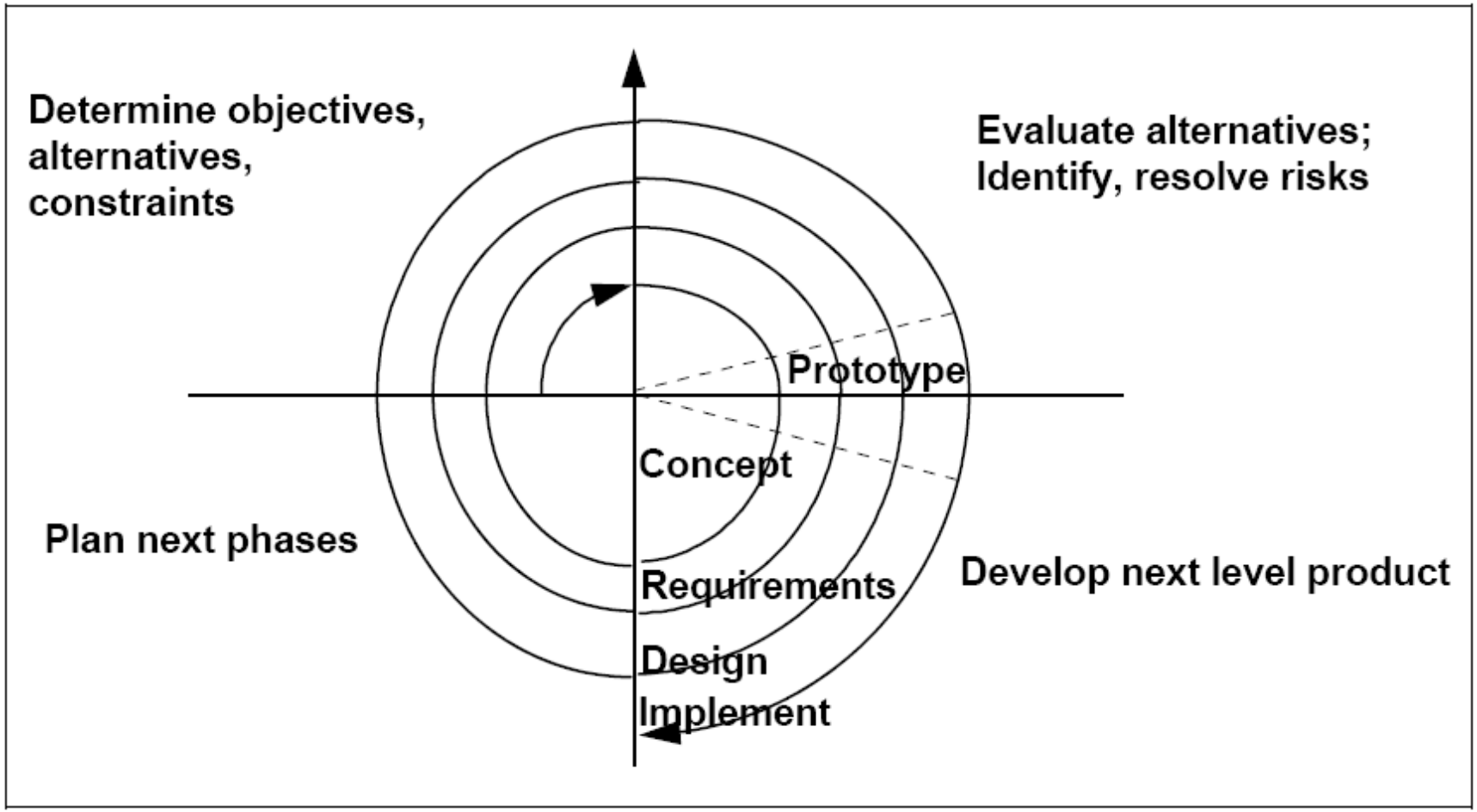
- Software engineering
- Phases for Developing Applications
- Developing Models
- Requirement Engineering

# From Course 1



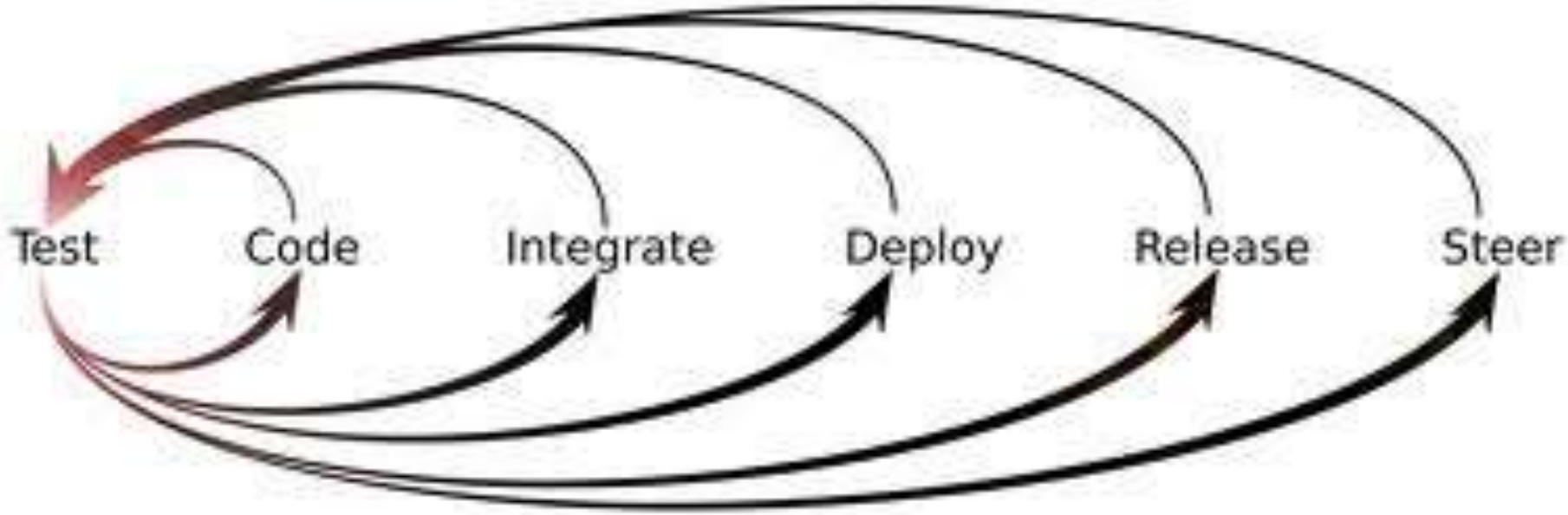

Waterfall model...

# From Course 1




Spiral model

# From Course 2




- XP, TDD…

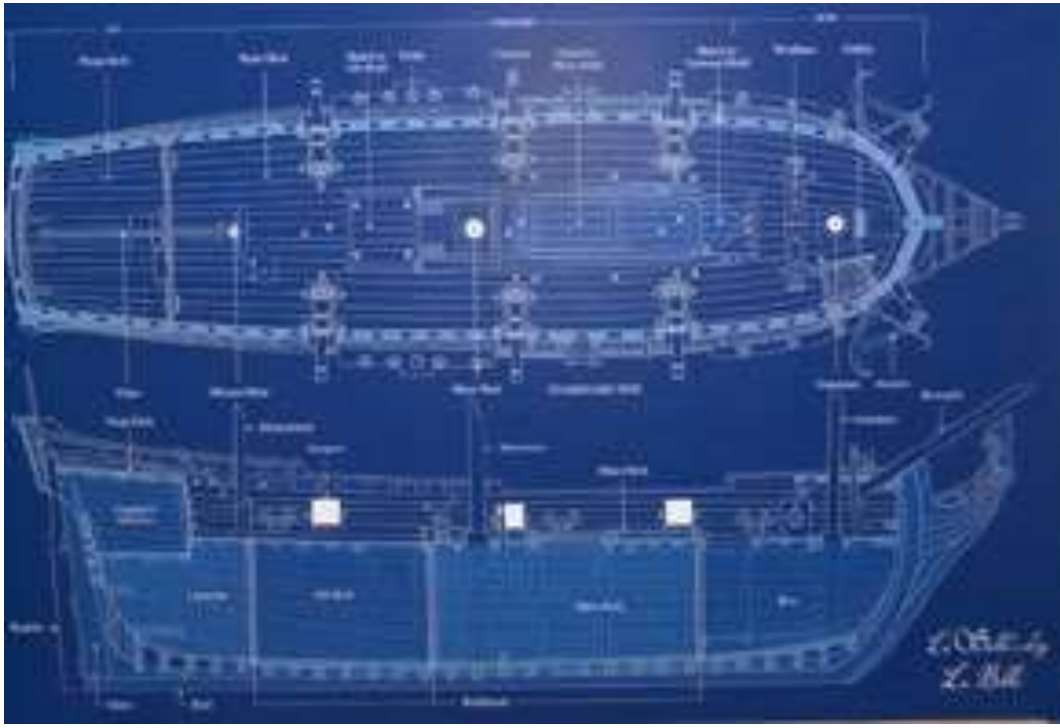
# From Course 2

- Requirement Engineering:
  - Actors
  - Scenarios

- Case Study
  - Actors identification
  - Scenarios identification (Obiective, Steps, Special cases)
  - Class Diagrams

# Modeling – Why?

- What is a model?
  - simplification of reality
  - A detailed plan system (blueprints)



- Why do we model?
  - To understand better what we do
  - To focus on one issue at a time
- Where do we use modeling?

# Modeling purposes

- Viewing a system
- Specifying its structure and / or behavior
- Providing a template to assist in building
- Documentation of decisions

# Modeling Architecture

- Using **Use cases**: to present requirements
- With the help of **Design**: capture vocabulary and problem domain
- Using **Process**: capture processes and threads
- With the help of **Implementation**: we model the application
- Using **Deployment**: capturing the system from an engineering point of view

# Modeling Principles

- Models influence the final solution
- You can use different levels of precision
- Good models have correspondent in reality
- One model is not enough

# Modeling Languages

- The analysis and the design of a project must be done **before the implementation of the code**
- Currently, companies give special attention to this stage, since the **production and the reuse of software** depends on it
- **Modeling languages** were **created** for the analysis and design of programs
- A modeling language is an **artificial language** that can be used to express information or knowledge or systems

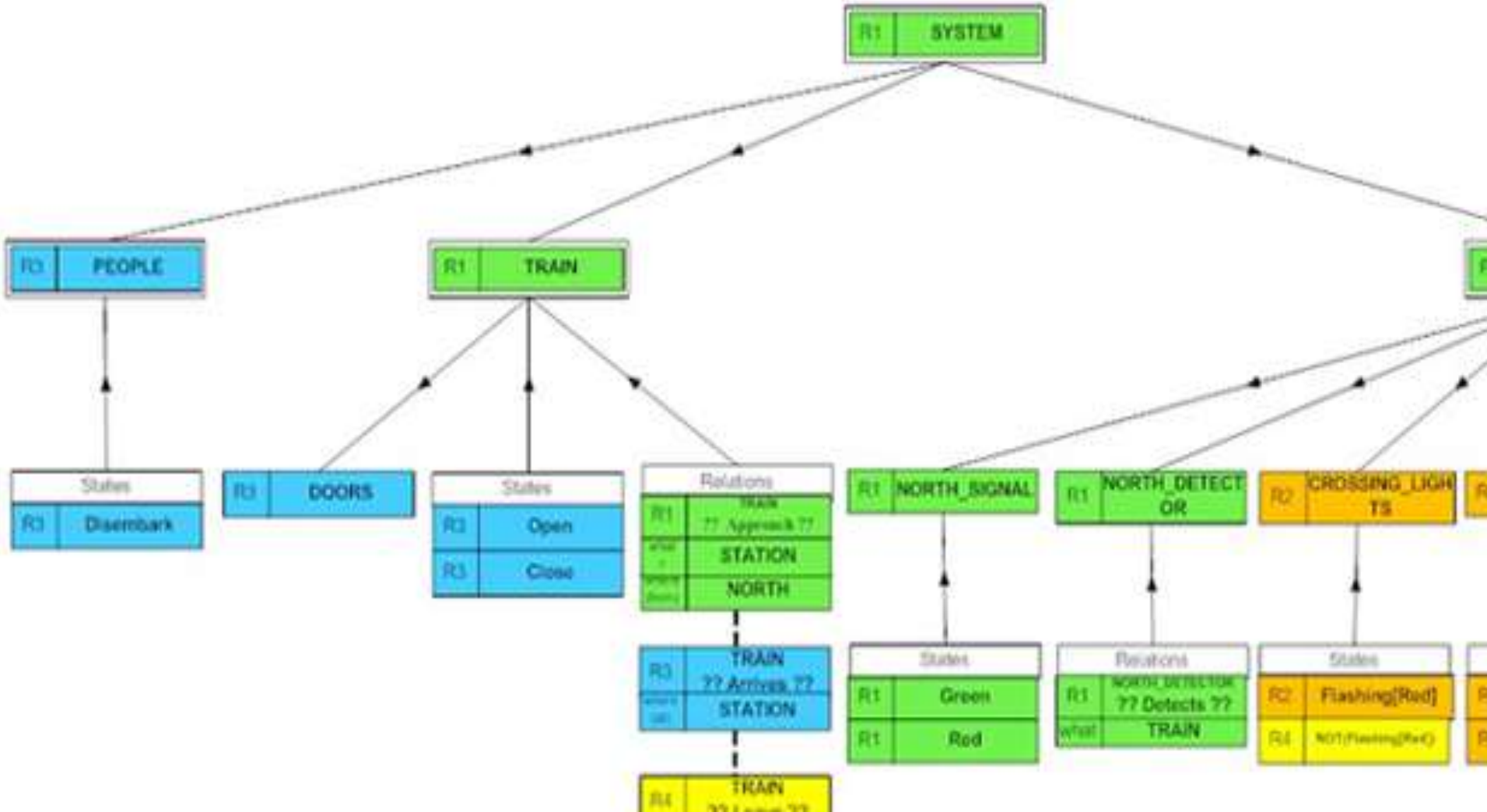
# Modeling Languages – Types

- **Graphic languages**: behavioral trees, business modeling language, EXPRESS (data modeling), flowchart, ORM (roles modeling), Petri nets, **UML diagrams**

- **Specific languages:** algebraic modeling (AML), domain specific languages (DSL), architecture modeling (FSML), object modeling language, virtual reality modeling (VRML)

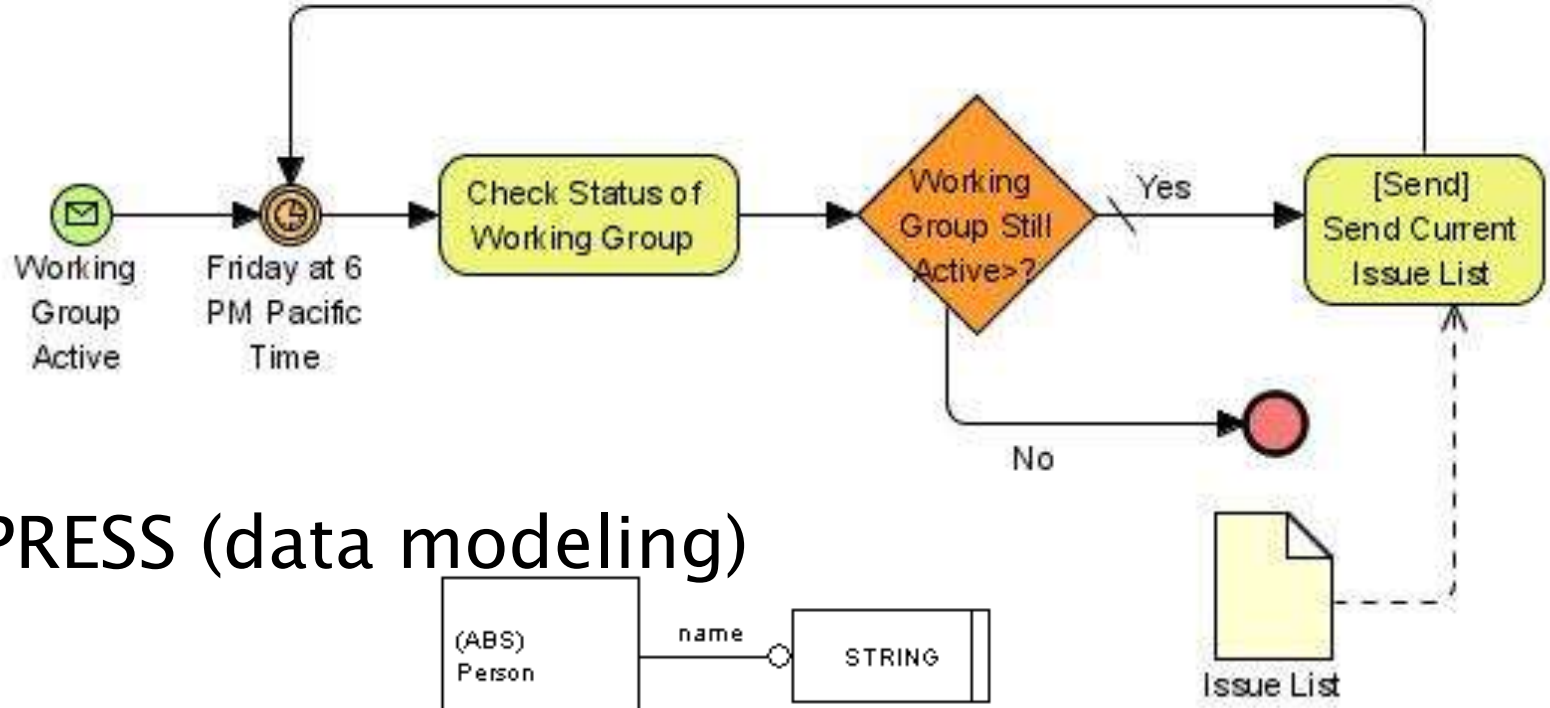
# Graphic Languages 1

- Behavioral trees

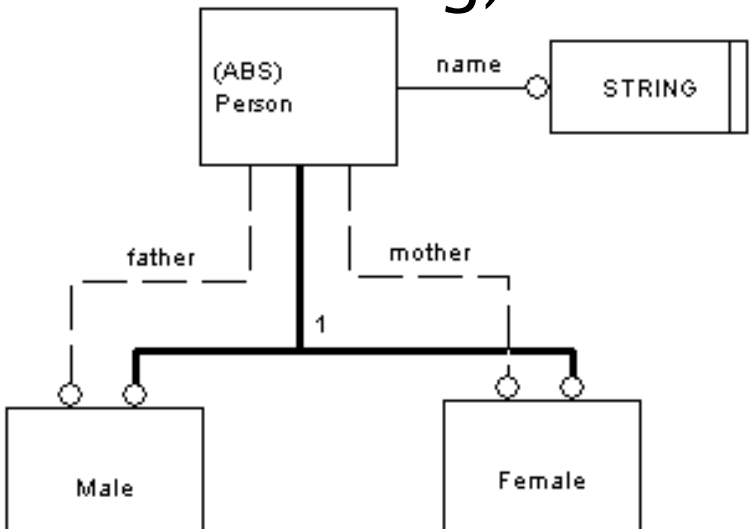


# Graphic Languages 2

- Business processes

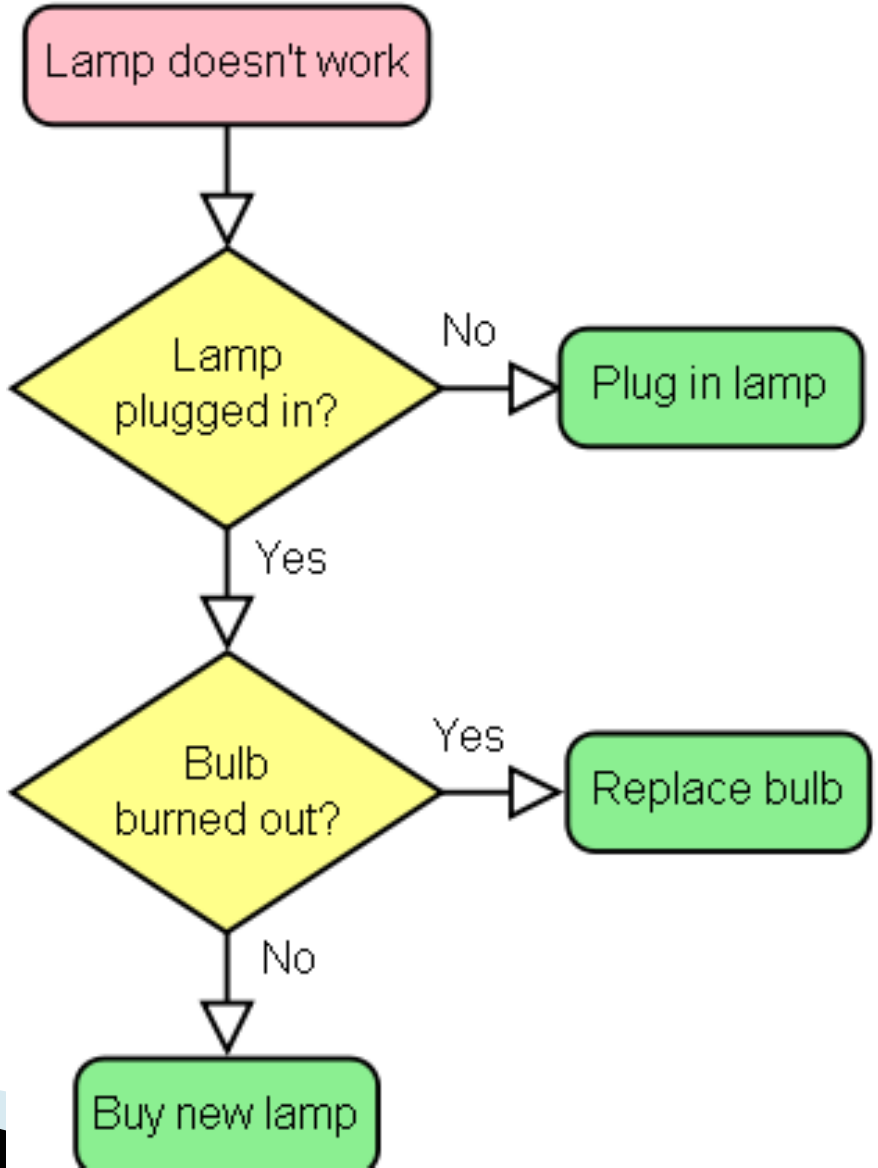



- EXPRESS (data modeling)



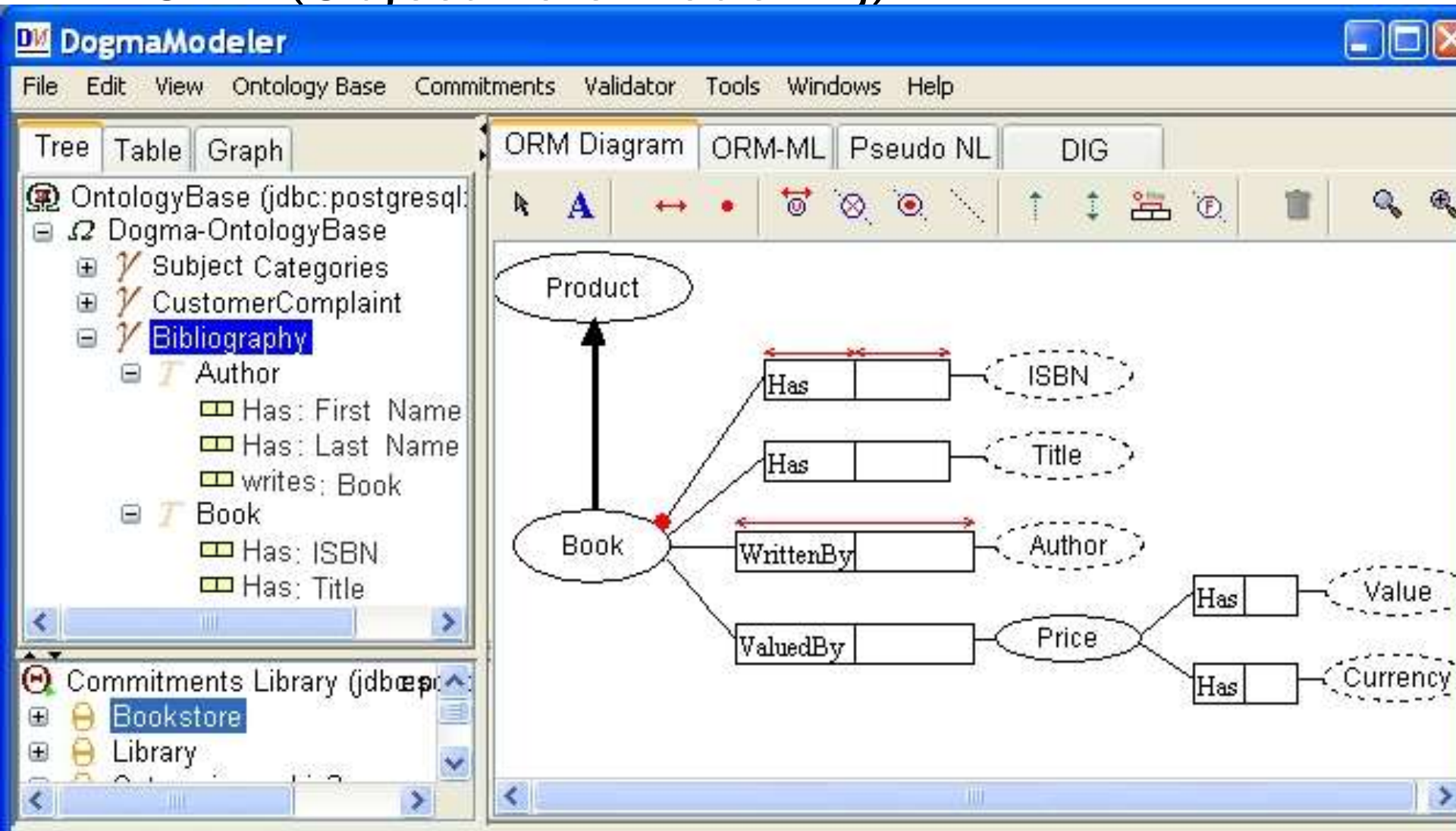
# Graphic Languages 3

- Flowchart



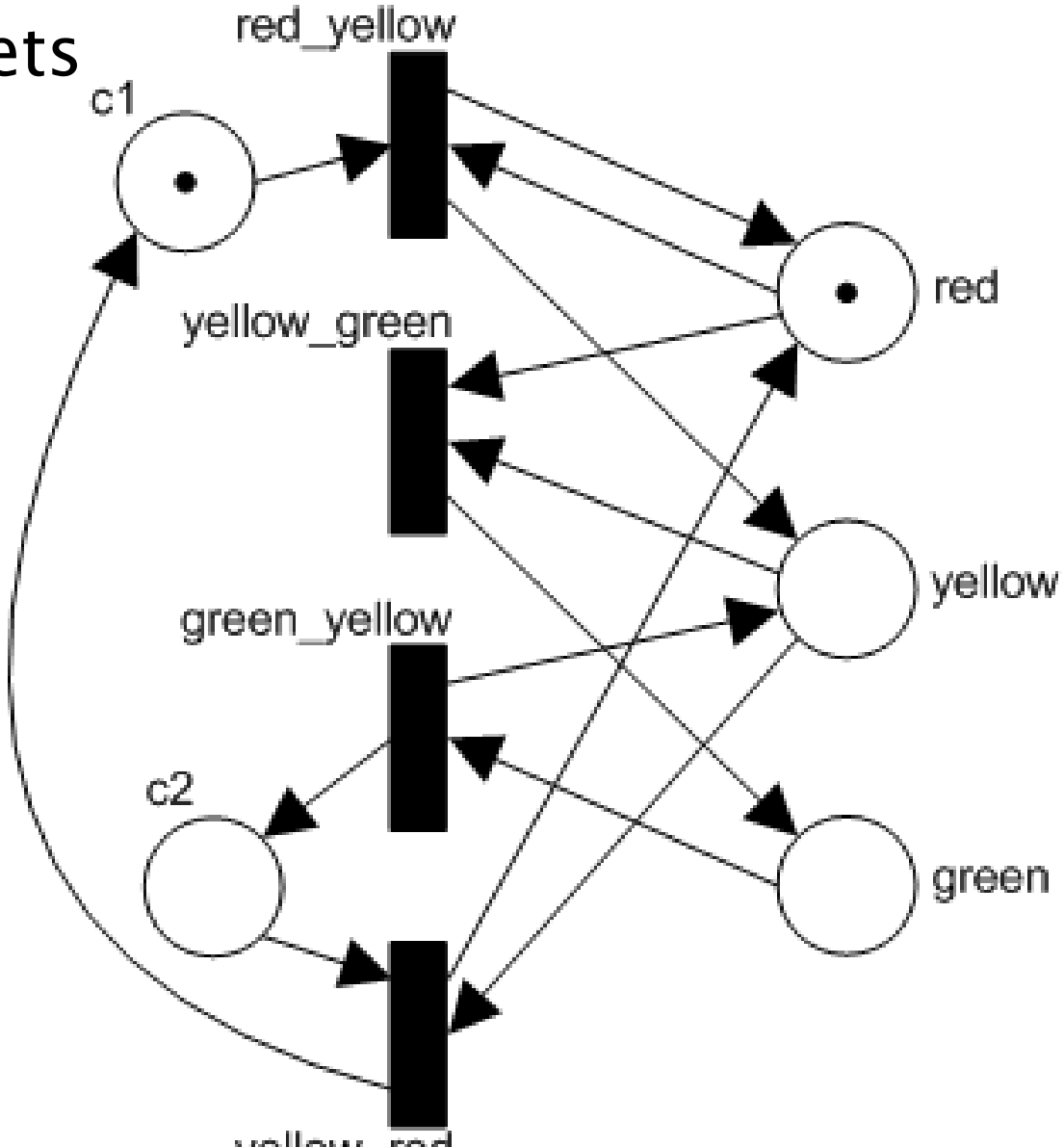
# Graphic Languages 4

- ORM (Object Role Modeling)



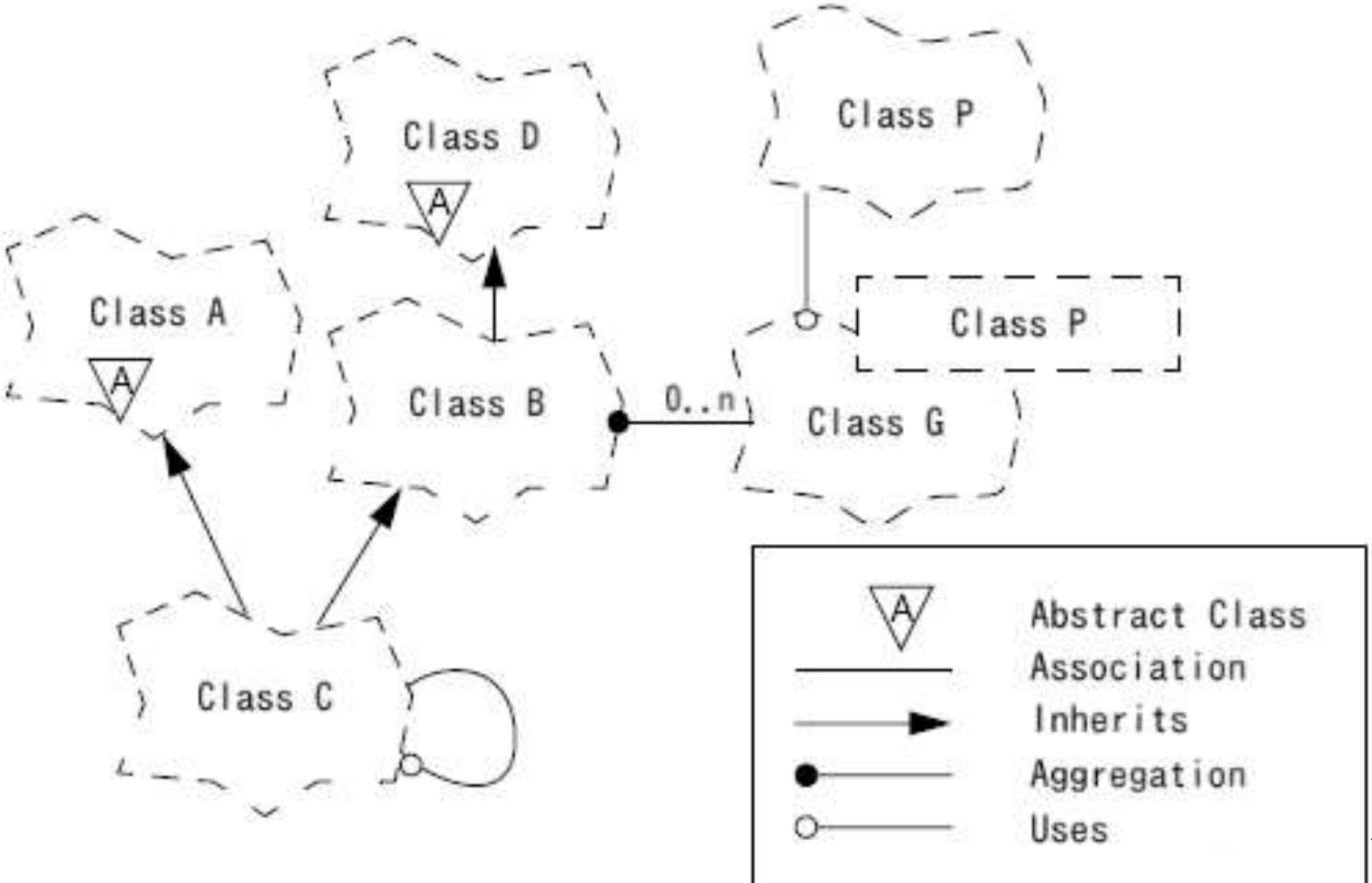
# Graphic Languages 5

- Petri Nets



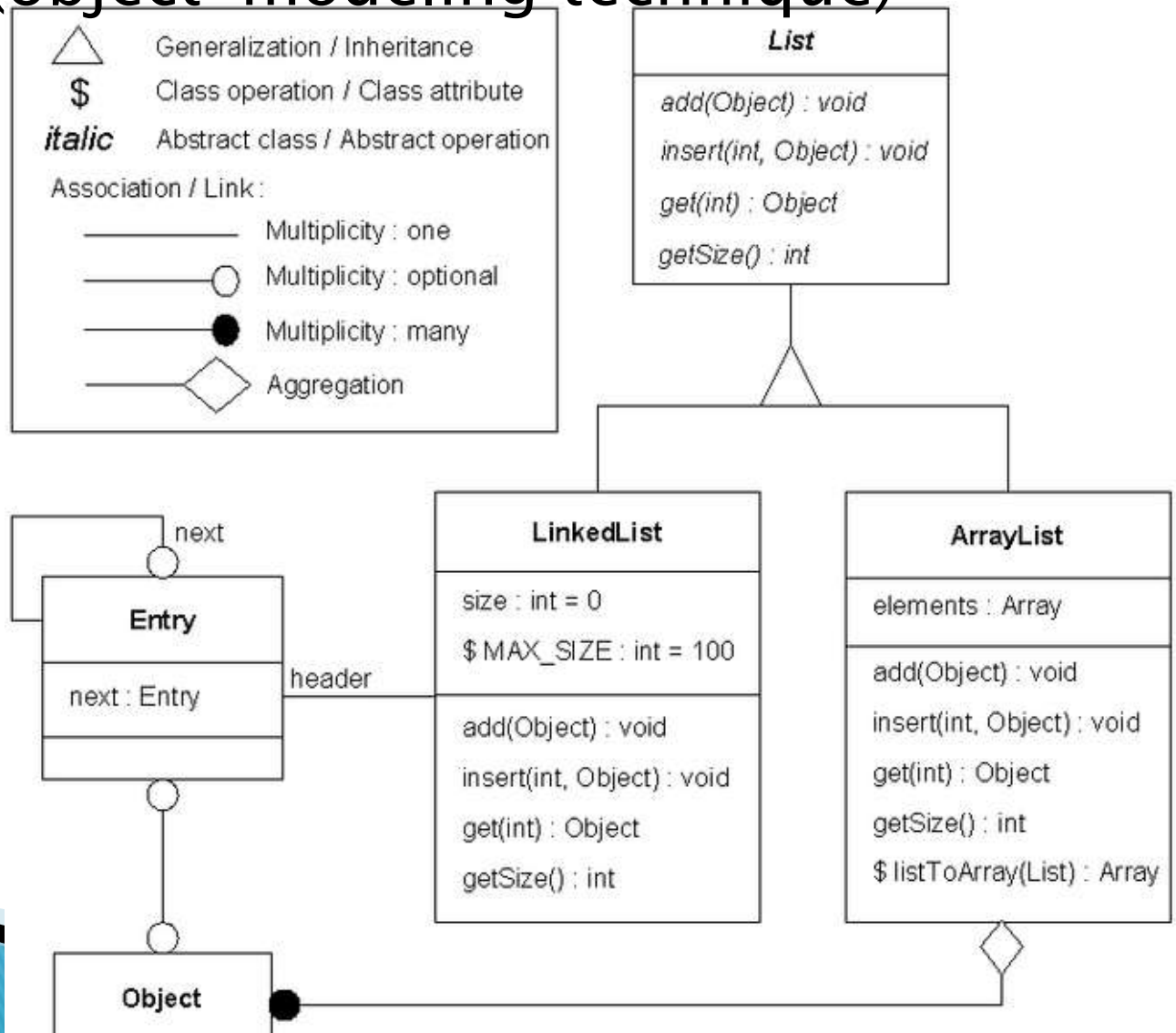
# Graphic Languages 6

- Booch Method (Grady Booch) – analysis and oo design



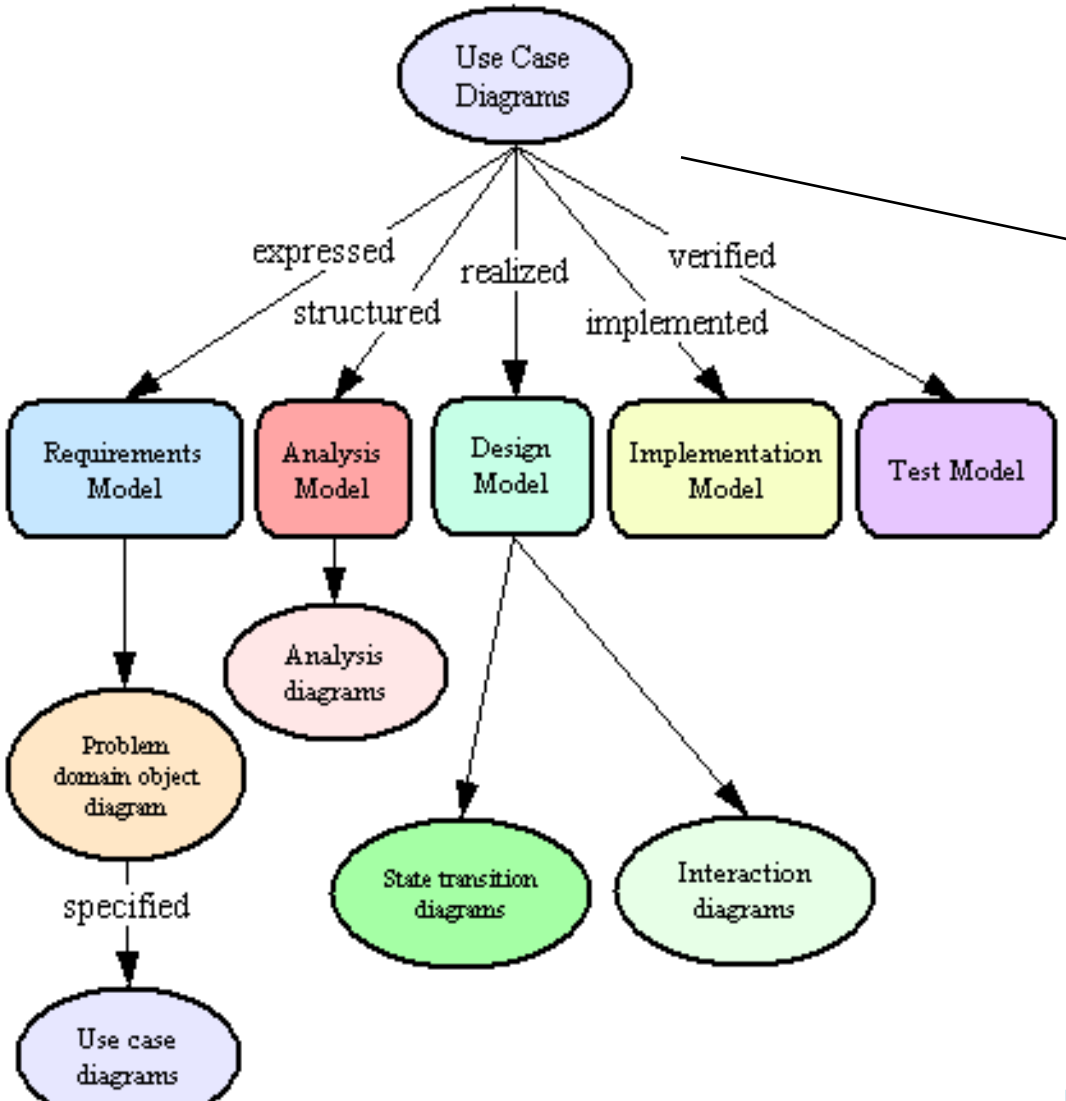
# Graphic Languages 7

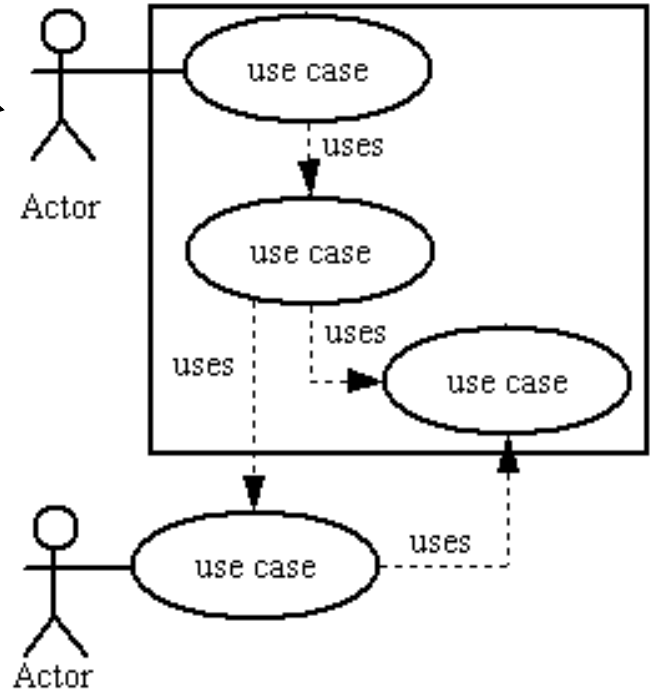
- OMT (object-modeling technique)



# Graphic Languages 8

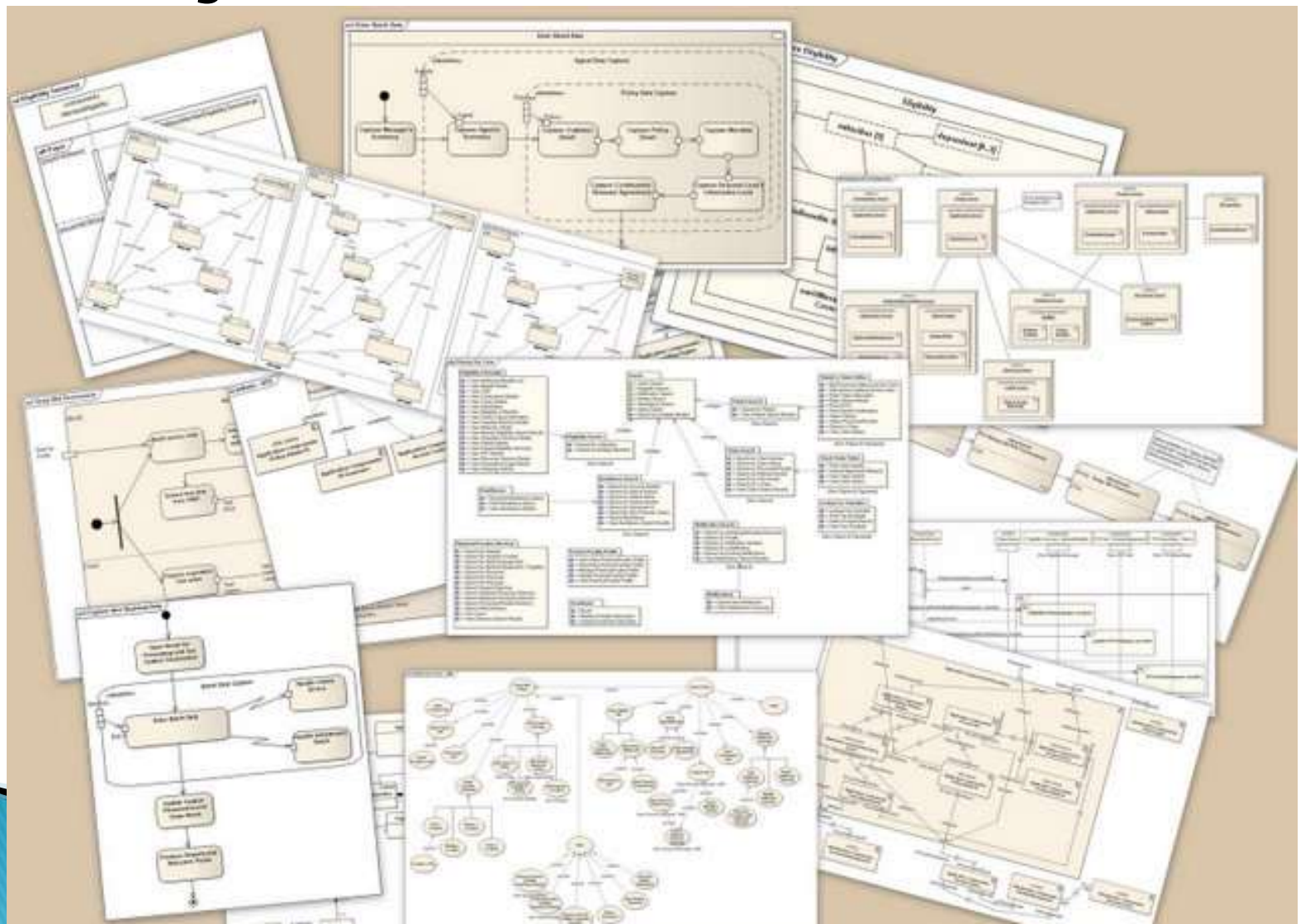
- OOSE (Object-oriented software engineering)



# Graphic Languages 9

- UML Diagrams

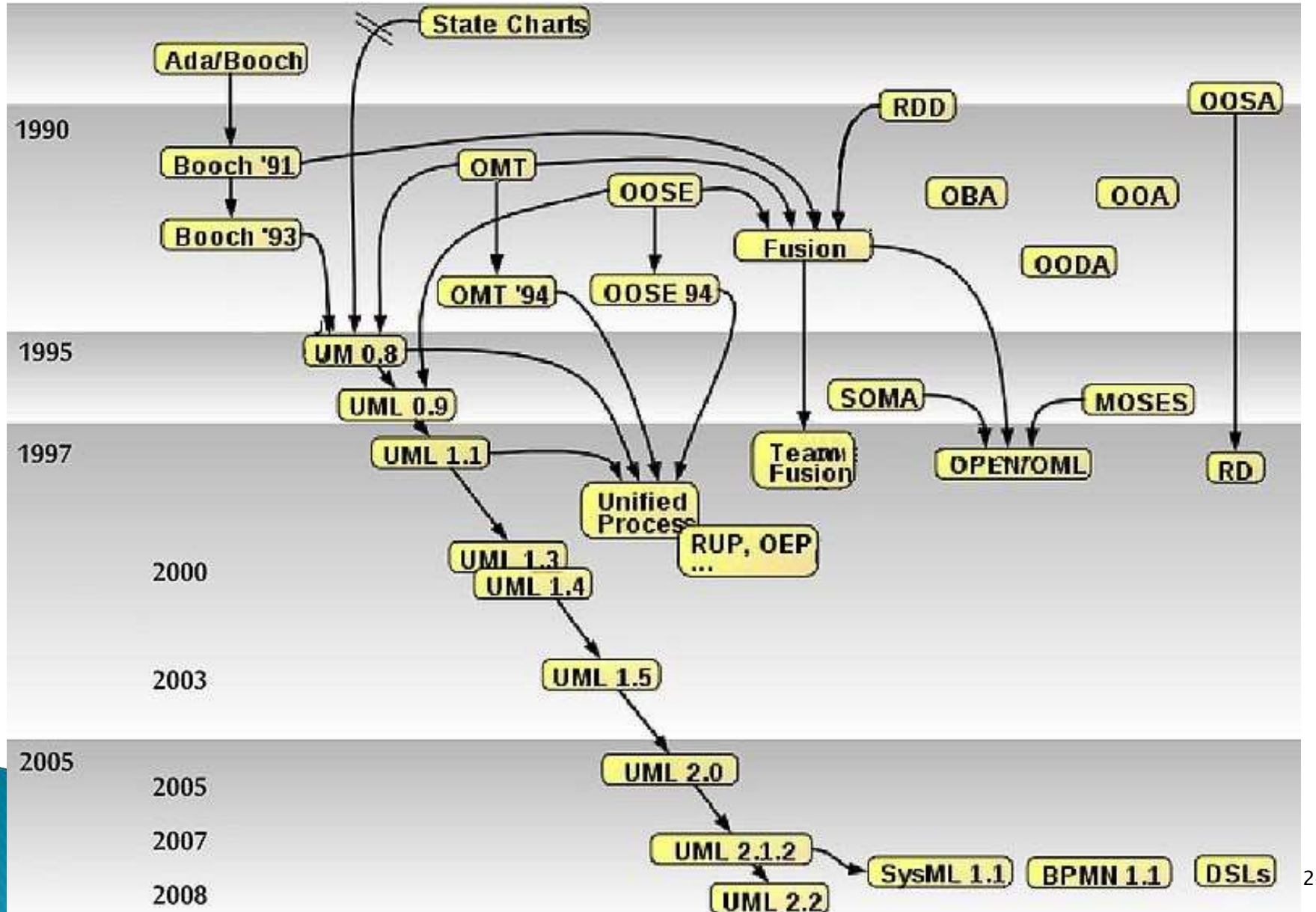


# UML – Introduction

- UML (Unified Modeling Language) – it is the successor of the three best OO modeling languages :
  - Booch (Grady Booch)
  - OMT (Ivar Jacobson)
  - OOSE (James Rumbaugh)

- UML consists of the union of these modeling languages and in addition it has a greater expressiveness

# UML – Evolution

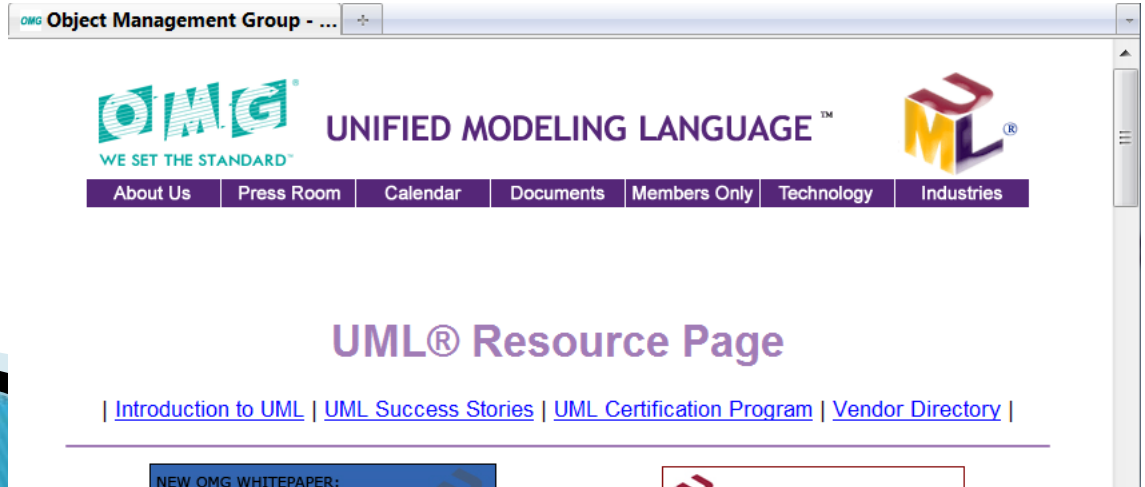


# UML – Definition (OMG)

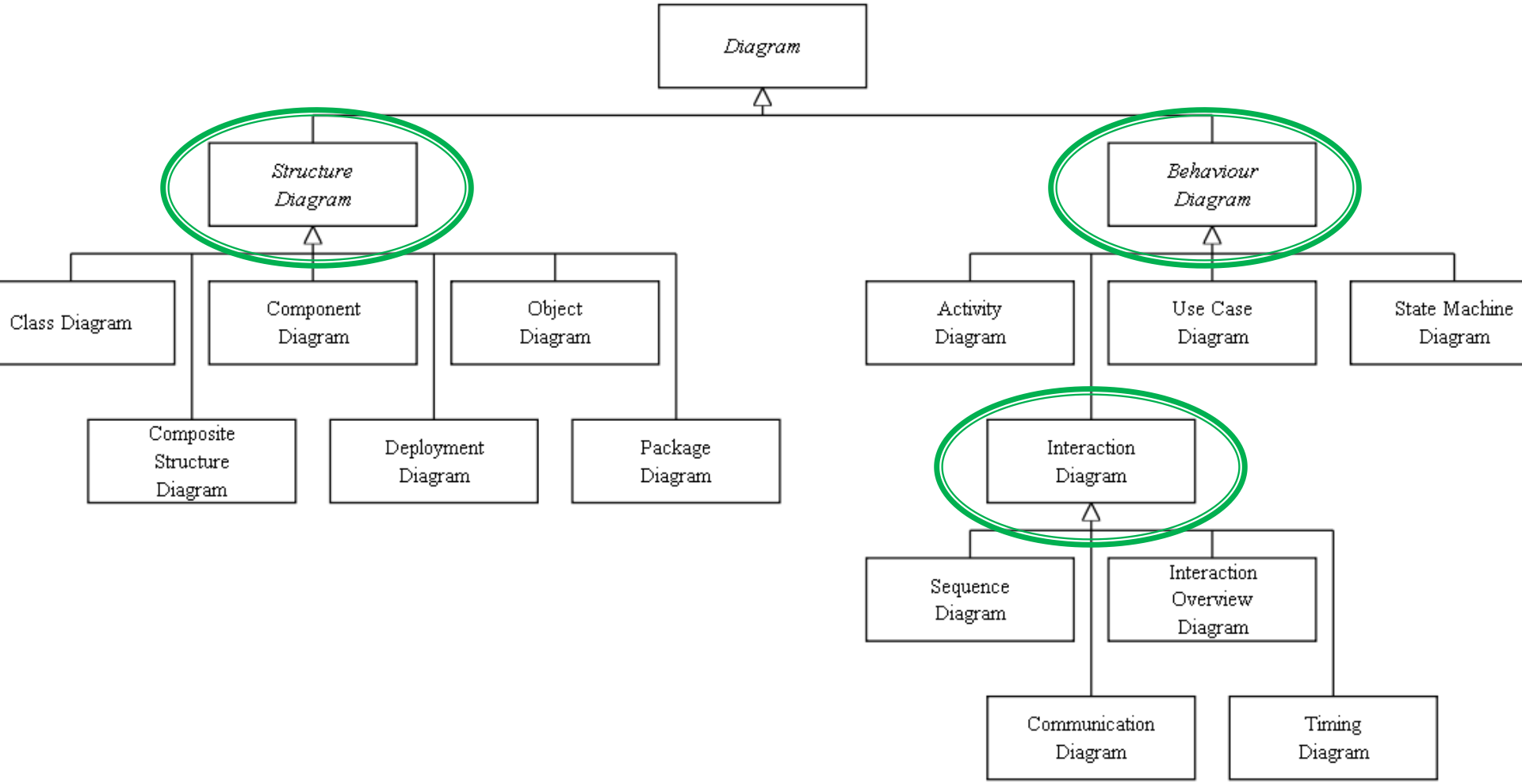
- *"The Unified Modeling Language (UML) is a* ***graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.***
- *The UML offers a* ***standard way to write a system's blueprints****, including conceptual things such as* ***business processes*** *and* ***system functions*** *as well as concrete things such as* ***programming language statements****,* ***database schemas****, and* ***reusable software components****."*

# UML – International Standard

- January 1997 – UML 1.0 was proposed for standardization by OMG (Object Management Group)
- November 1997 – Version UML 1.1 was adopted like standard by OMG
- Last version is UML 2.5
- Official site: http://www.uml.org

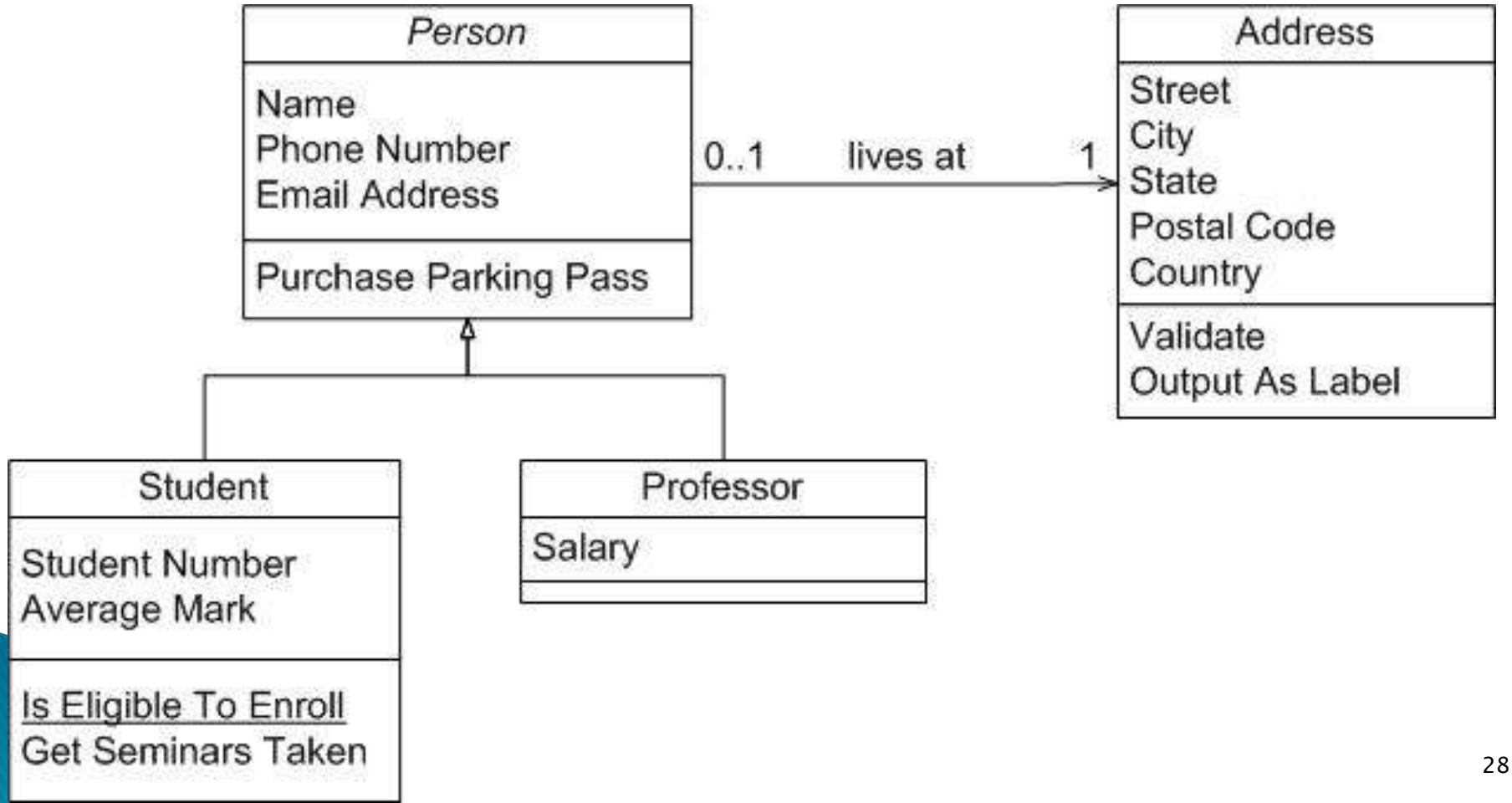


# UML2.0 – 13 Types of Diagrams



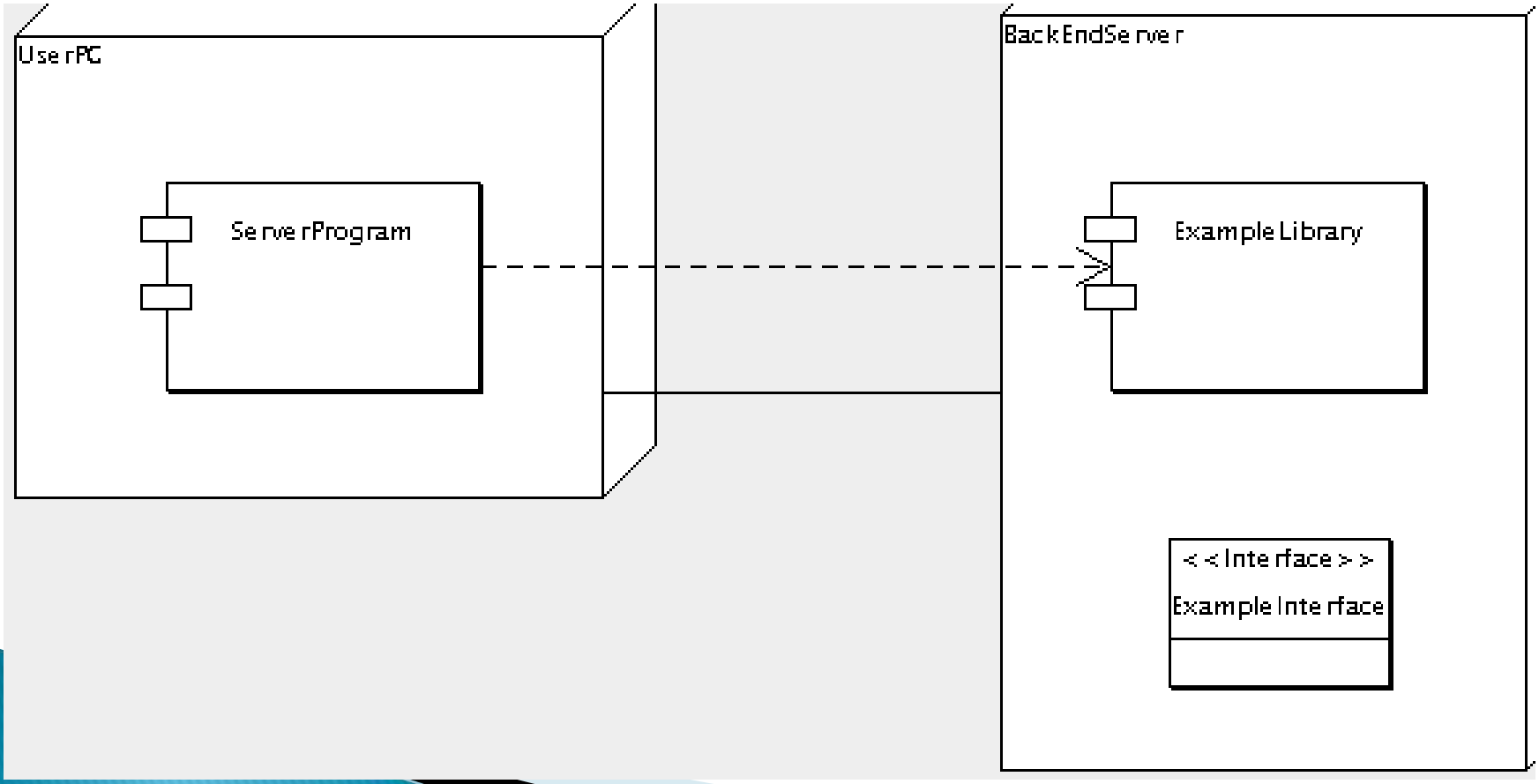
# UML2.0 – Structure Diagrams 1

- **Class Diagrams**: classes (attributes, methods) and relations between classes



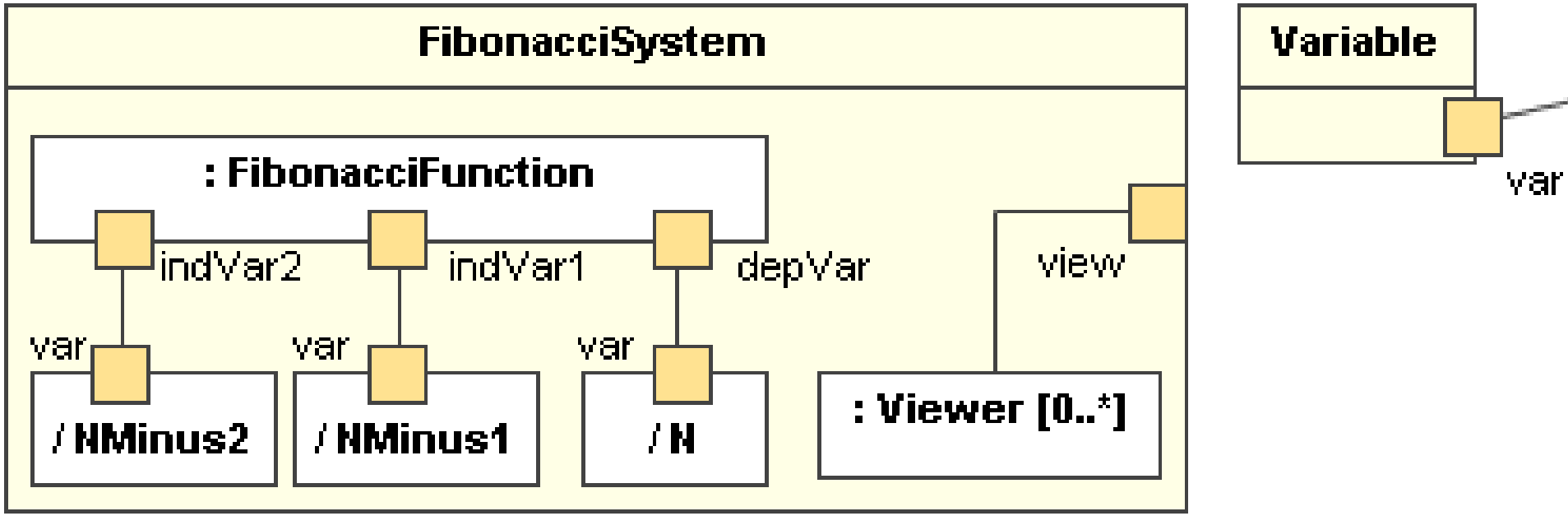
# UML2.0 – Structure Diagrams 2

- **Component Diagrams:** main components and relations between them



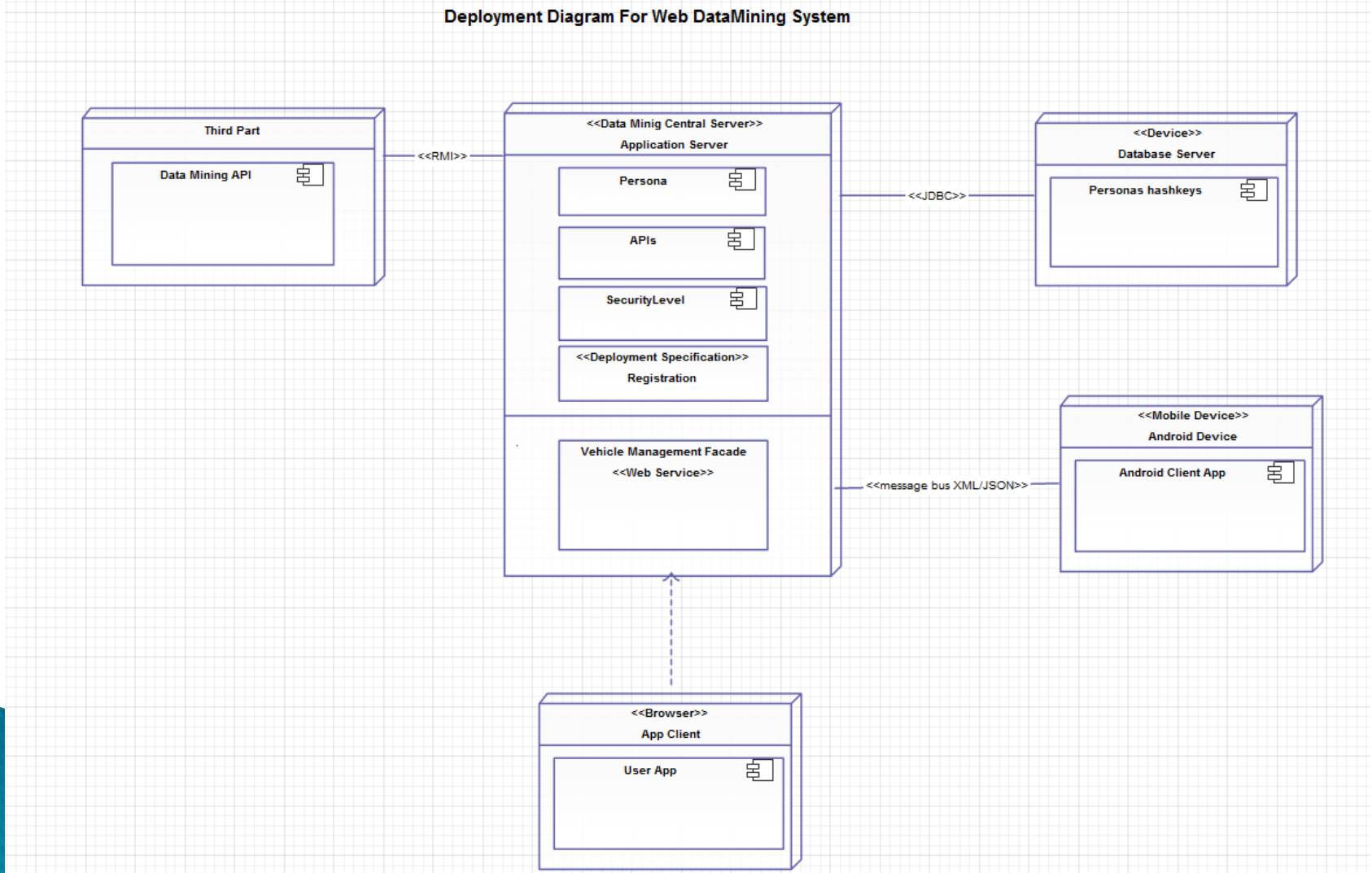
# UML2.0 – Structure Diagrams 3

- **Composite structure diagrams**: intern structure



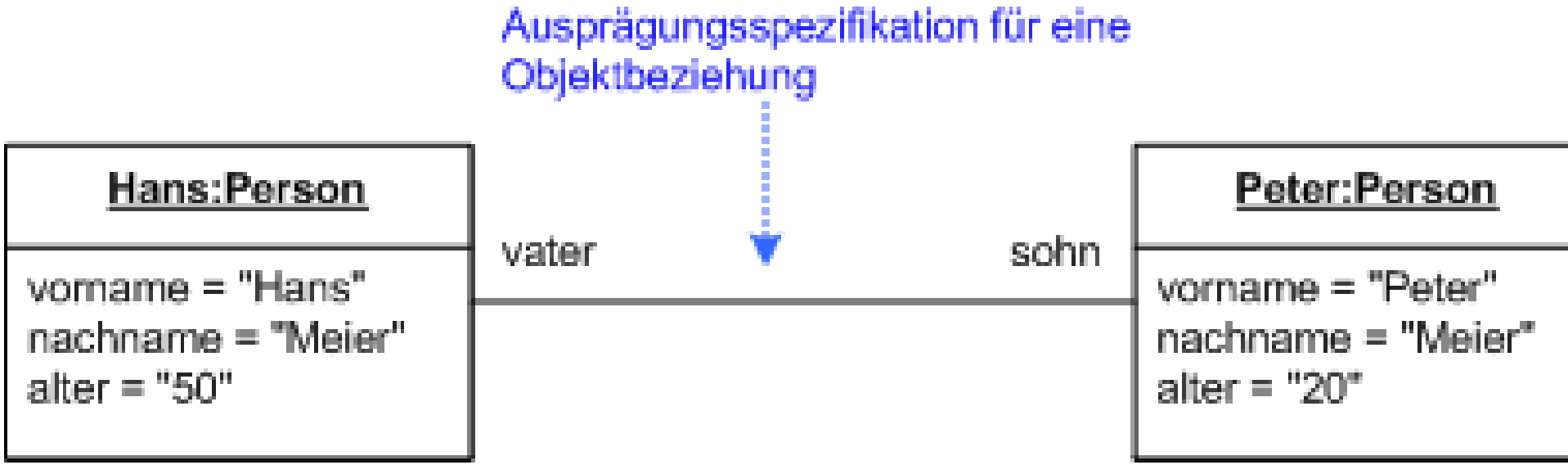
# UML2.0 – Structure Diagrams 4

- **Deployment Diagram**: hardware structure



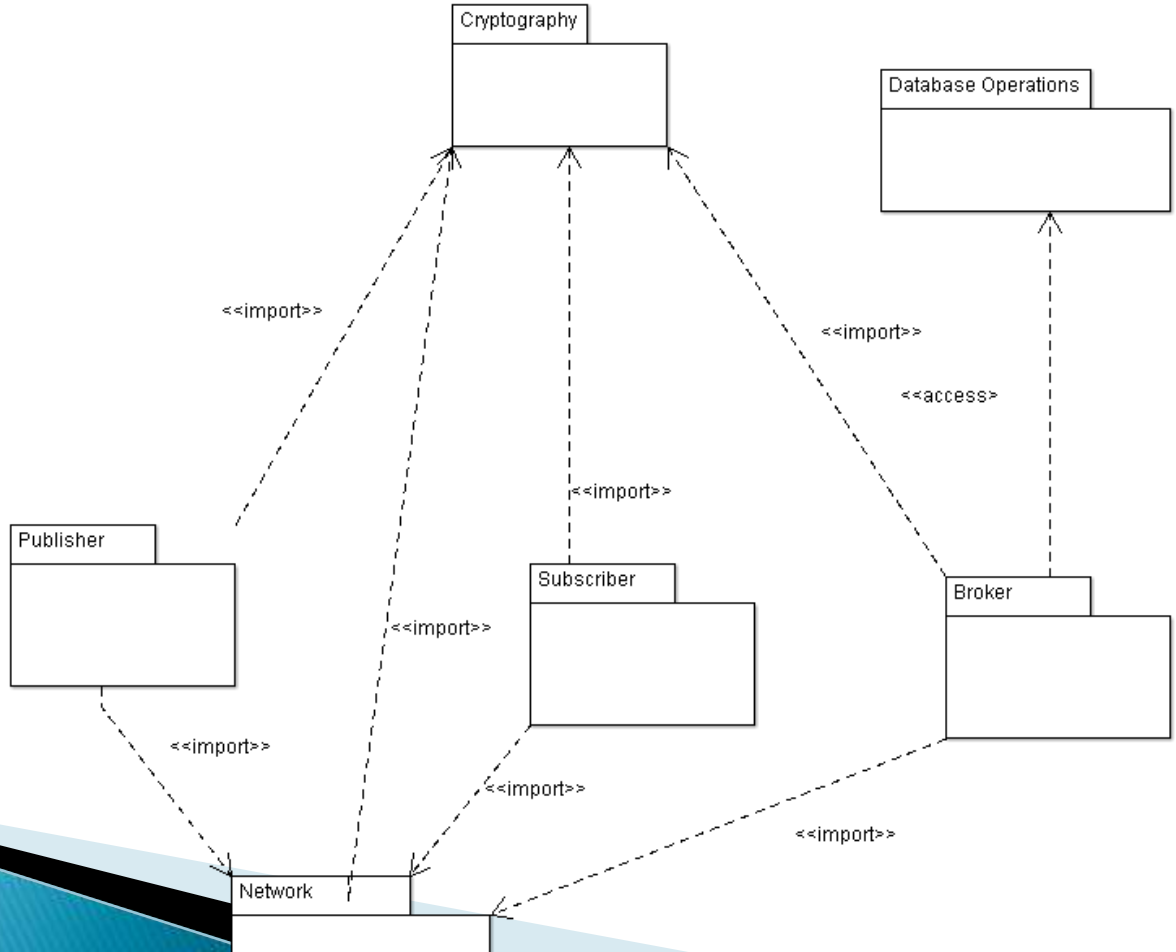
# UML2.0 – Structure Diagrams 5

- **Object Diagrams**: system structure at a given time



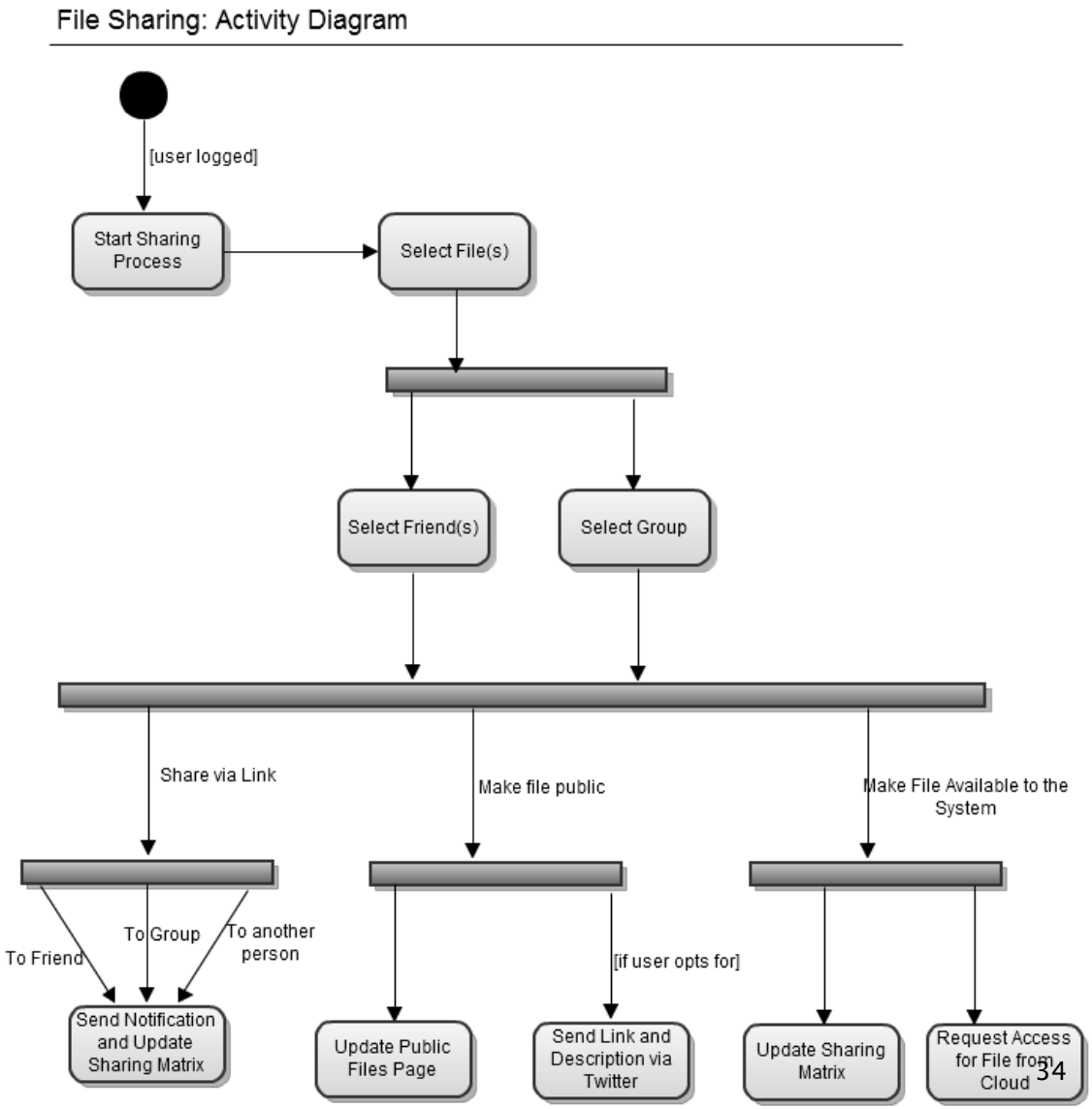
# UML2.0 – Structure Diagrams 6

- **Diagram package:** the system is divided into packages with the relationships between them



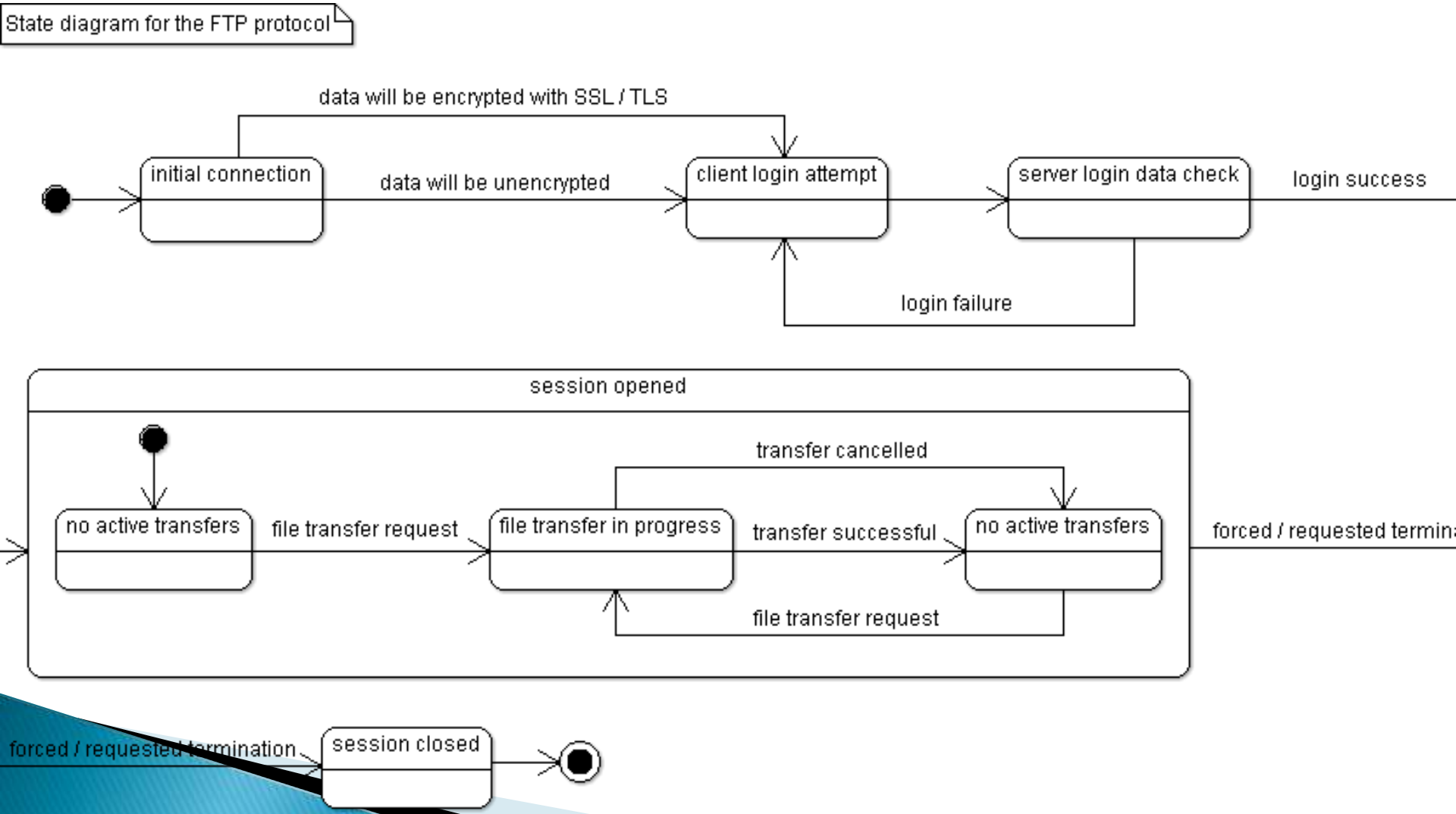
# UML 2.0 – Behavioral Diagrams 1

- **Activity diagrams:** business presentation and workflow



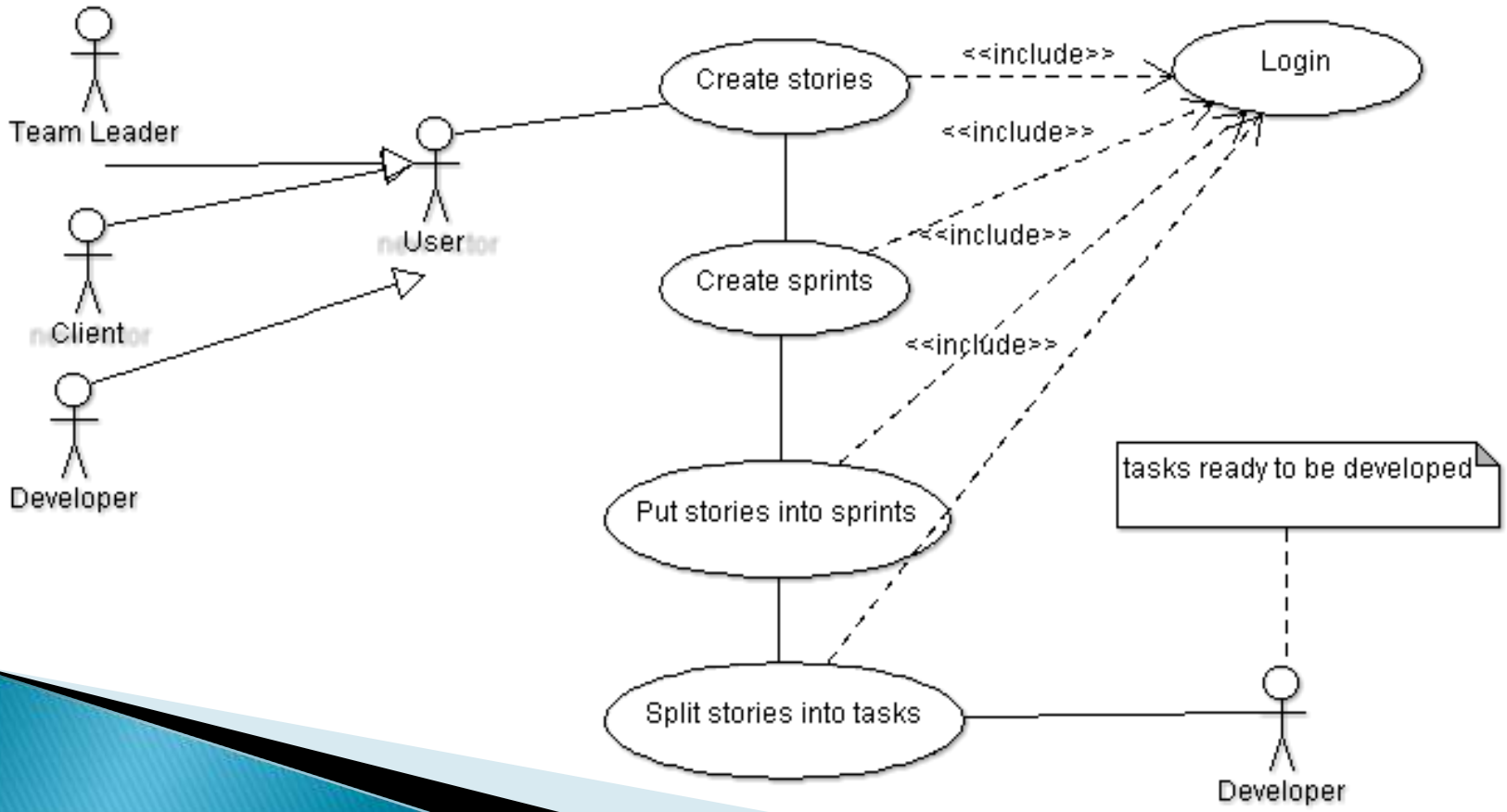
# UML 2.0 – Behavioral Diagrams 2

- **State diagrams:** to present states of objects



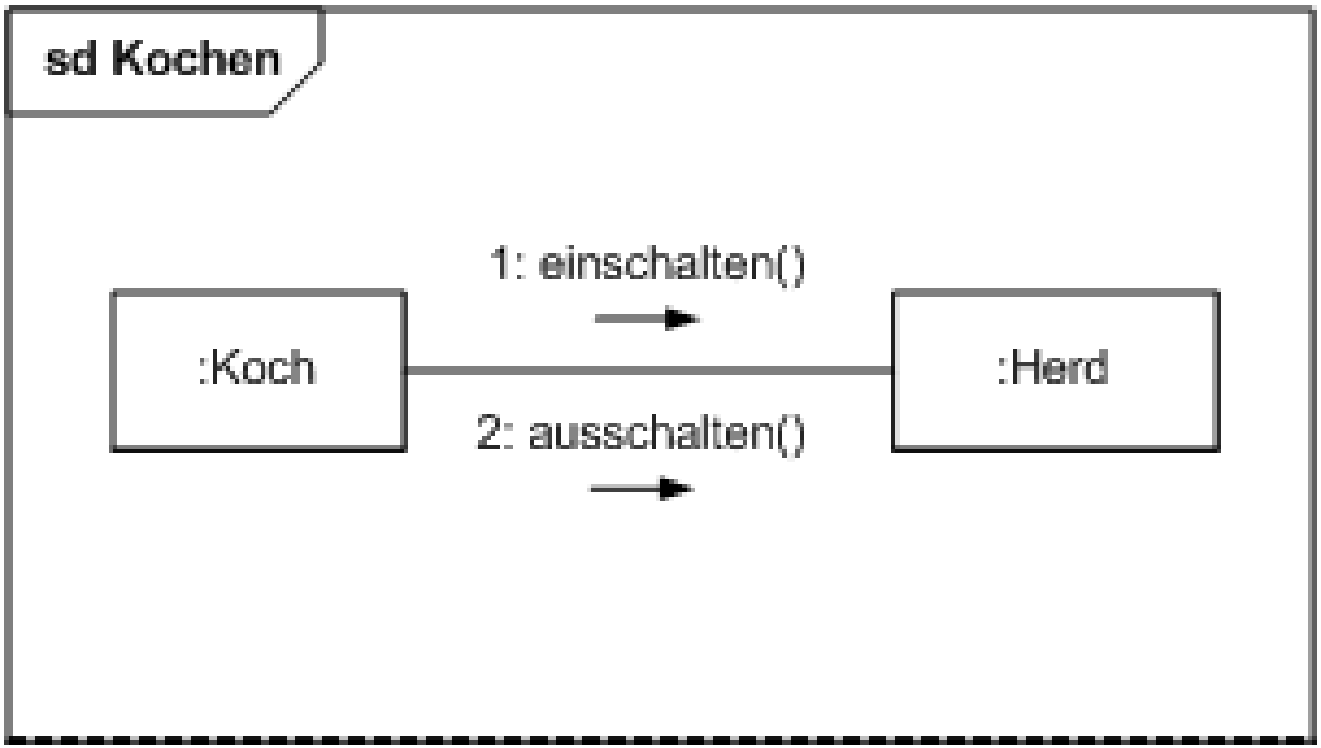
# UML 2.0 – Behavioral Diagrams 3

- **Use Case Diagrams**: show the functionality of the system using actors, use cases, and dependencies between them



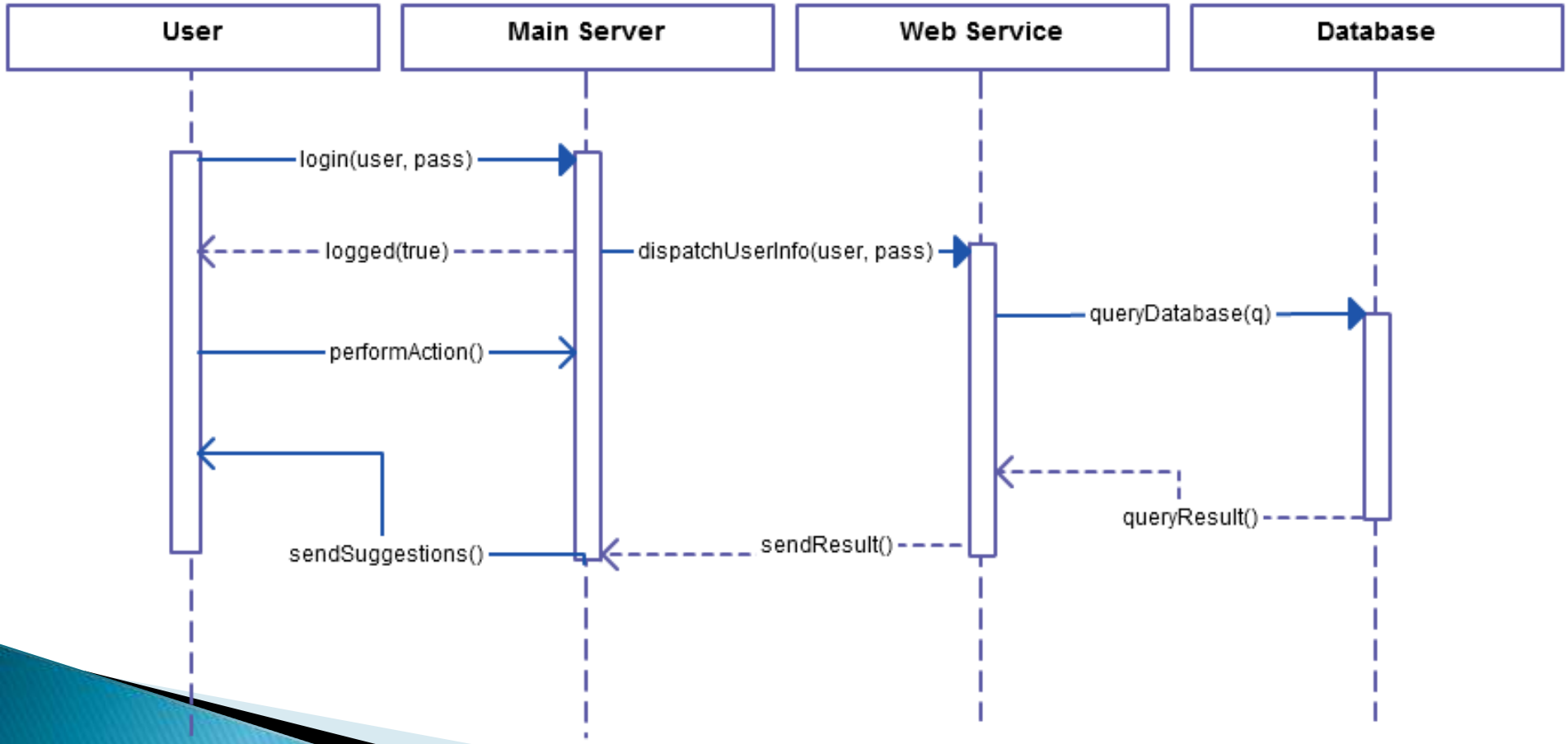
# UML 2.0 – Interaction Diagrams 1

- **Communication diagram:** shows the interactions between objects (the dynamic behavior of the system) (actors: chef, stove, actions: cooking, firing, disconnection)



# UML 2.0 – Interaction Diagrams 2

- **Sequence Diagram:** shows how objects communicate with each other in terms of sending messages



# Use Case Diagram

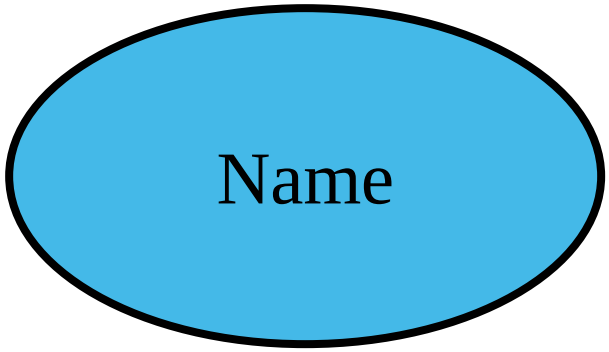
- It is a behavioral diagram that captures **system requirements**
- Delimiting **borders of the system**
- The starting point is the document obtained after requirements engineering
- Can present:
  - **requirement specification** (external) from the user's point of view
  - Specification of the **system functionality** in terms of the system
- Contain:
  - **Use Cases** = functionalities of the system
  - **Actors** = external entities with which the system interacts
  - **Relations**

# Use Case

- It is a description of a **set of sequences of actions** (including variants) that executes a program when **interacting with external entities** (actors) and **leading to a noticeable result**
- It can be a system, a subsystem, a class, a method
- It is a **program functionality**
- Specify **what makes** a program or subprogram
- It **does not specify how to implement a functionality**
- The identification of a Use Case is being done starting with the **customer requirements and problem description**

# Use Case – Representation

- Notation
- Attributes
  - Name = **verbal phrase** that denotes an **operation** or a **behavior** from the area of the problem
- Restrictions
  - The name is unique



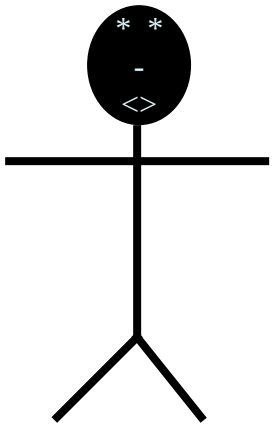
# Actor

- Is a **role** that users play when interacting with an Use Case
- It is an entity outside of the system
- Interacts with the system:
  - **Initiate execution** of use cases
  - **It provides functionality** for the realization of the use cases

- It can be:
  - Human
  - Software component
  - Hardware component

# Actor – Representation

- Notation
- Attributes
- Name = **indicate the role** which the actor plays in interacting with an Use Case
- Restrictions
  - The name is unique



# Relations

- Established between two elements
- Relation types:
  - **Association:** Actor – UseCase, UseCase – UseCase
  - **Generalization:** Actor – Actor, UseCase – UseCase
  - **Dependence:** UseCase – UseCase (<<include>>, <<extend>>)

# Association Relation

- Modeled communication between the elements that connect them

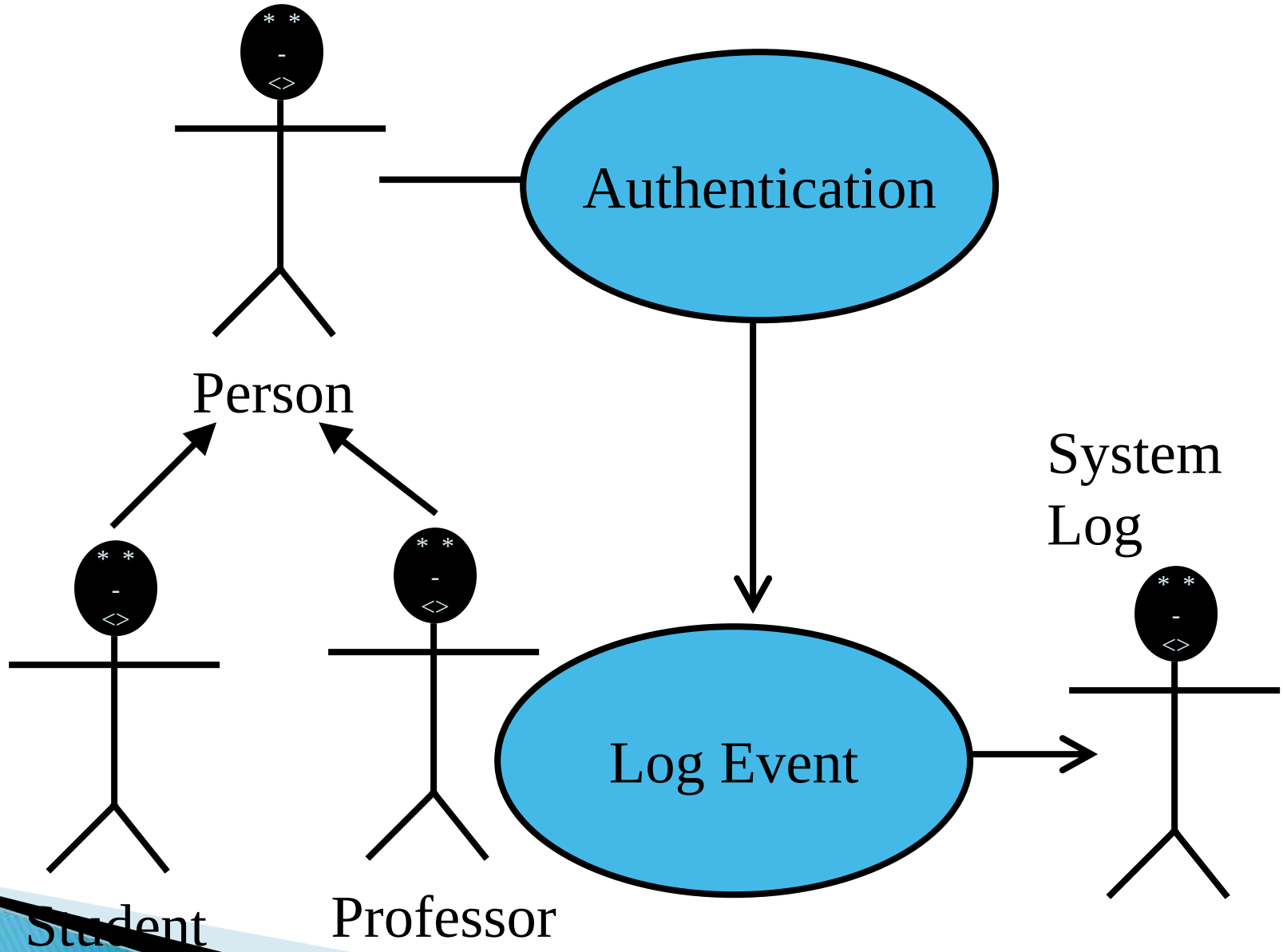
- Can appear between:
  - **An actor and an UseCase** (An actor initiates the execution of a use case or it provides a functionality for its realization)
  - **Two Use Cases** (data transfer, sending messages/ signals)

- **Notation** _____

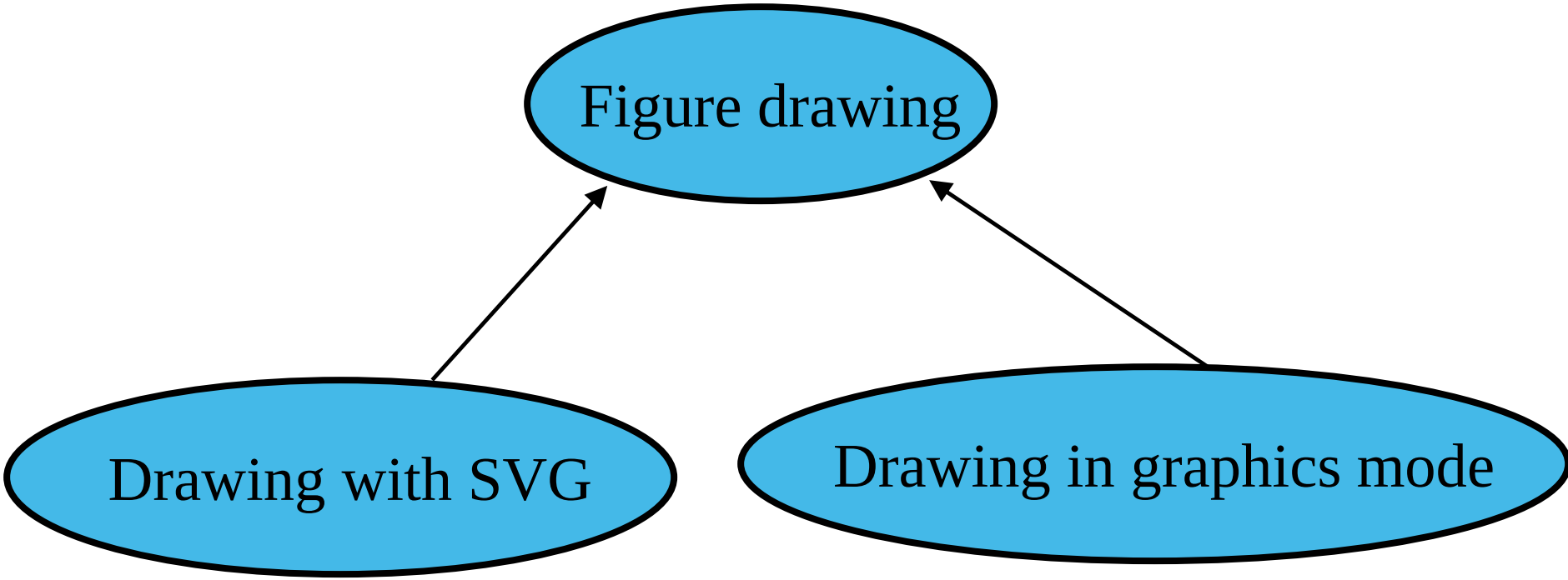
# Generalization Relation

- **It is between elements of the same type ⇒ hierarchies**
- It models the situations in which an item is a special case of another element
- The particular element inherits the relations in which the general element is involved in
- **Notation:** ←

# Example 1


Person
Authentication
System
Log
Log Event
Student
Professor

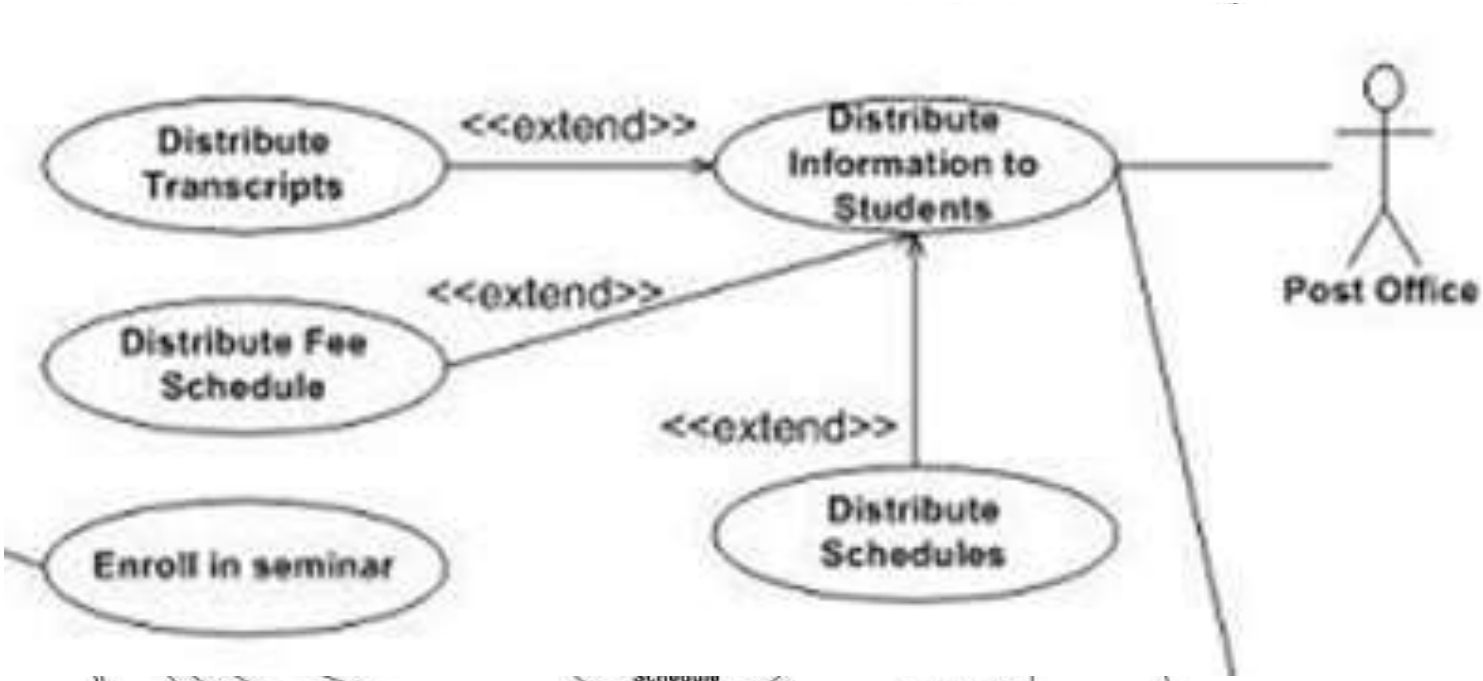
# Generalization

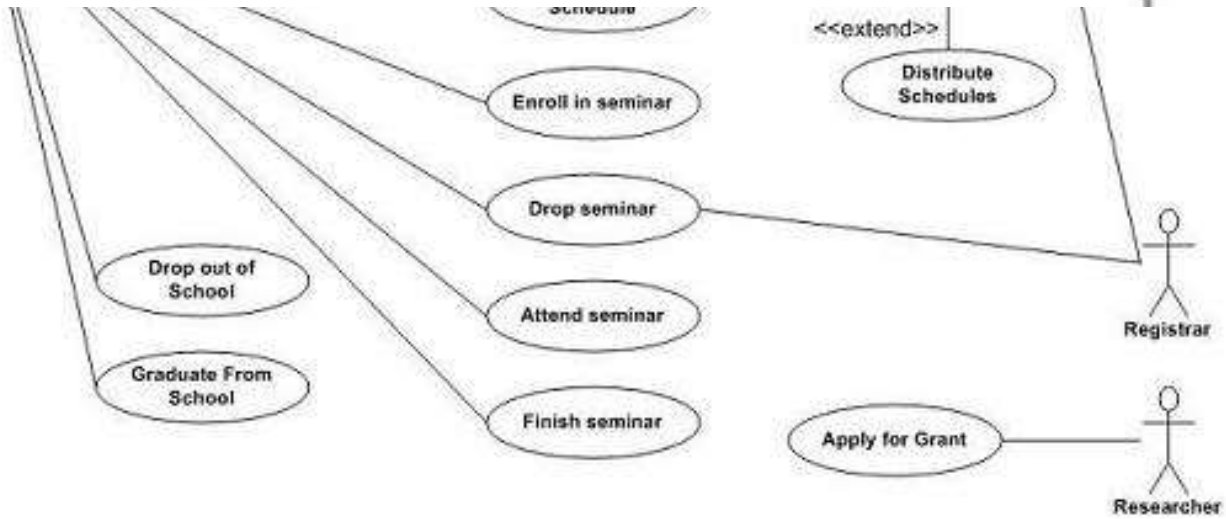


# Dependence

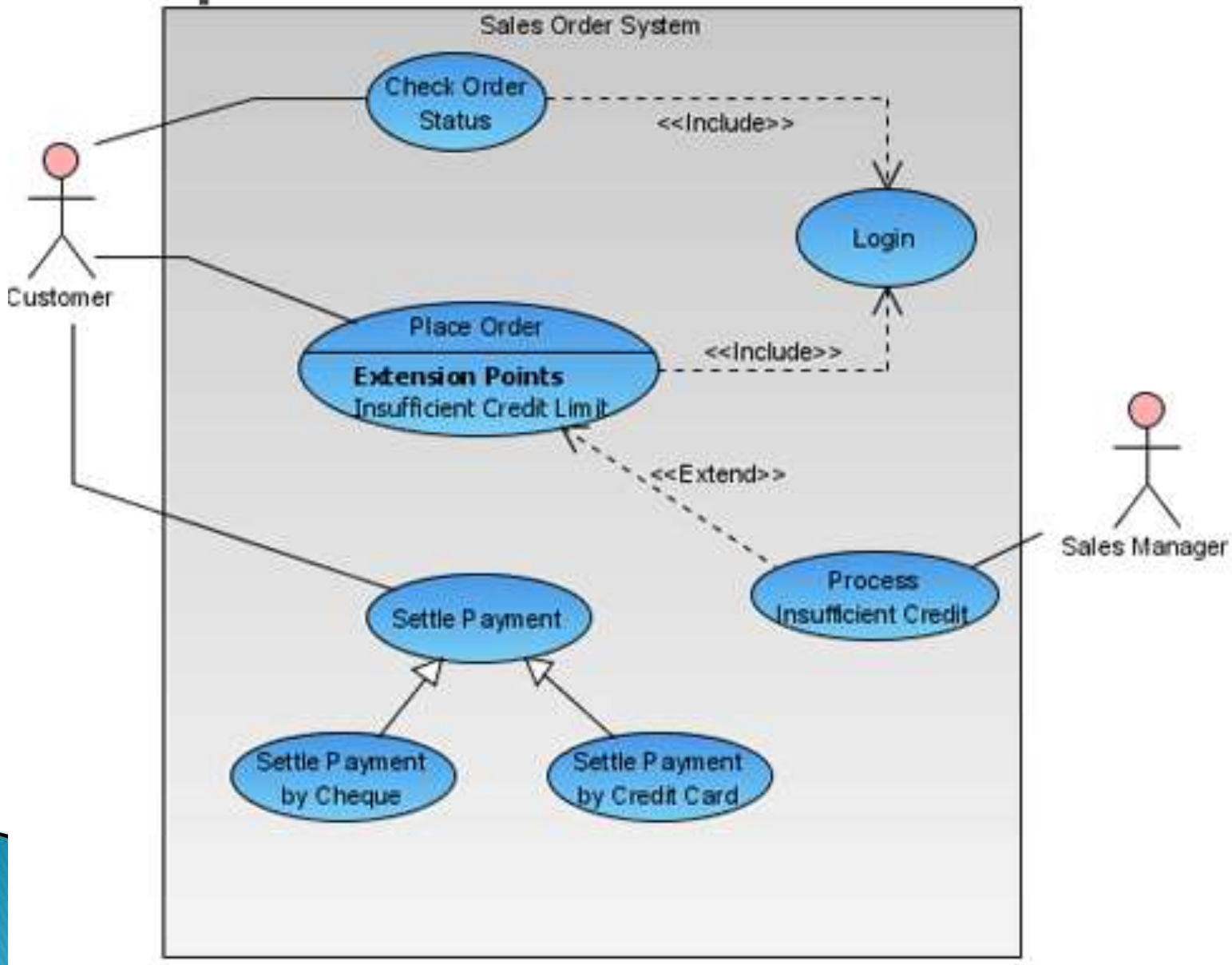
- Between two Use Cases
- Modeling situations in which
  - An Use Case uses the behavior from another Use Case (<<include>>)
  - The behavior of a Use Case can be extended by another Use Case (<<extend>>)
- **Notation** <- - - - - - - - - - - - - - -

# Example 2

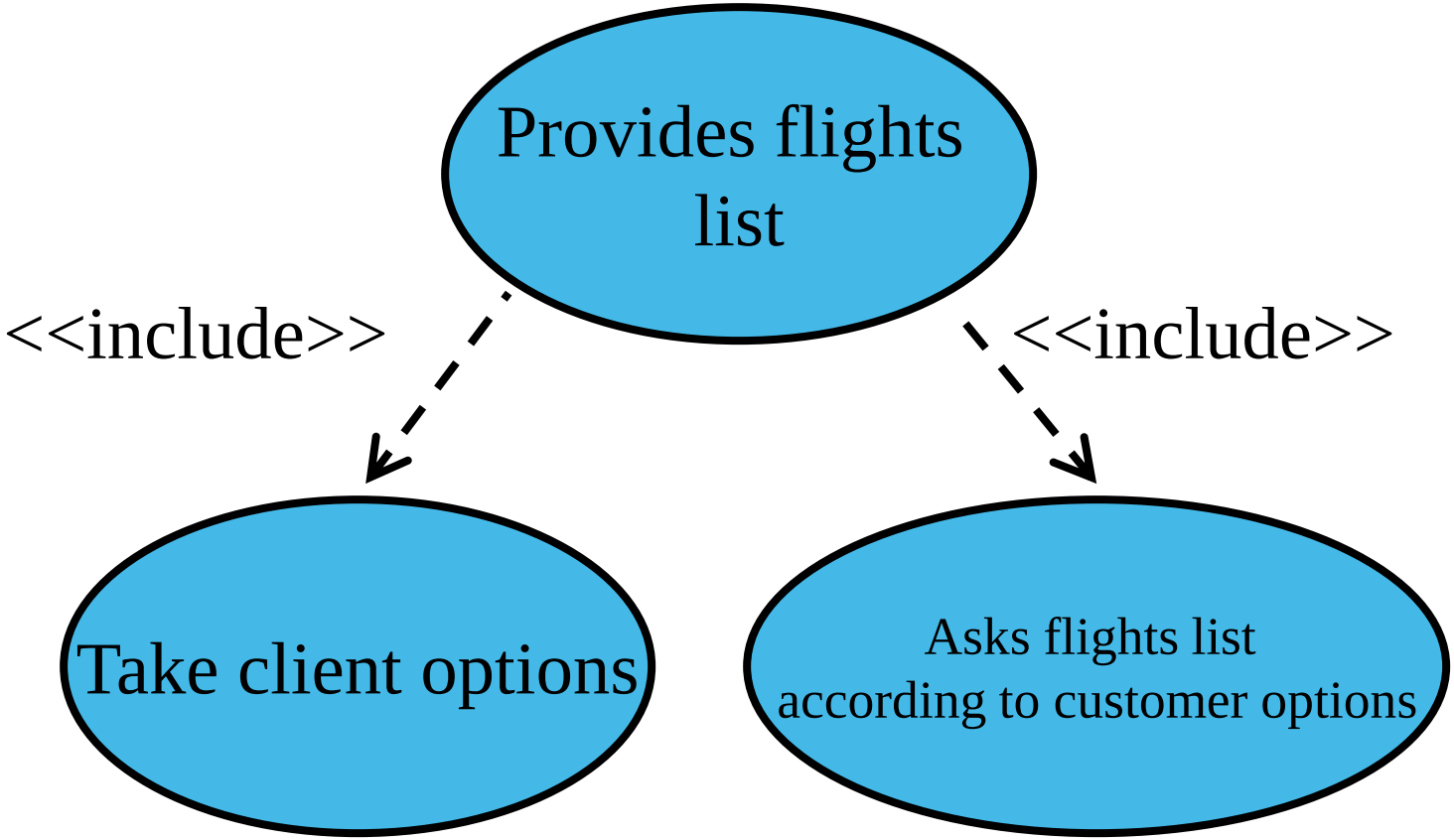

Distribute Transcripts
<<extend>>
Distribute Information to Students
Post Office
<<extend>>
Distribute Fee Schedule
<<extend>>
Distribute Schedules
Enroll in seminar


<<extend>>
Distribute Schedules
Enroll in seminar
Drop seminar
Registrar
Drop out of School
Attend seminar
Graduate From School
Finish seminar
Apply for Grant
Researcher

# Example 3


Sales Order System
Check Order Status
<<Include>>
Login
Customer
Place Order
Extension Points
Insufficient Credit Limit
<<Include>>
<<Extend>>
Sales Manager
Process Insufficient Credit
Settle Payment
Settle Payment by Cheque
Settle Payment by Credit Card

# Example 4


Provides flights list
<<include>>
<<include>>
Take client options
Asks flights list according to customer options

# OO Concepts – Recapitulation

- **Object, Class, Instance**
- **Object:** Entity with: identity, state, behavior
  - Example: My yellow tennis ball with a diameter of 10 cm, which jumps
- **Class:** Description of a set of objects with the same structural characteristics, the same behavioral characteristics
  - Example: balls that have color, diameter, usage, jump
- **Instance**: an object belonging to a class
  - Example: Viorel Popescu is a Student

# OO 1

- It is any approach that includes
  - **Encapsulating data**
  - **Inheritance**
  - **Polymorphism**

- **Encapsulating data** (example Point class)
  - It means putting together data (attributes) and code (methods)
  - The data can be modified (only) by methods
  - Data hiding: we do not care how it provides services, but that it offers
  - If you change the layout or implementation, the interface remains unchanged

# OO 2

- **Inheritance**:
  - Some classes are specializations (customizations) of other classes
  - A subclass has (inherited) characteristics of super-class, it can expand in a certain way
  - An instance of a derived class is automatically an instance of the base class
  - Example (Student – Person)

- **Polimorfism**
  - Interpretation of semantics of a method call is made by the object who receives the call
  - Example: I tell a form: DRAW UP. It draws 4 lines if it is a square or it makes some points around the center, in case it is a circle
  - Also, I do not care who does the job or how it is done

# Class Diagram

- **Purpose:**
  - Modeled vocabulary of the system to be developed
  - Capture semantic connections and interactions that are established between components
  - Used to model the structure of a program
- **Contains**
  - Classes/Interfaces
  - Objects
  - Relations (Association, Aggregation, Generalization, Dependence)

# Classes

- Modeled vocabulary = identifies the concepts that the client or the programmer uses to describe the problem solution
- Elements of classes:
  - Name: identifies a class
  - Attributes: properties of the class
  - Methods: implementing a service that can be called by any instance of the class

Name

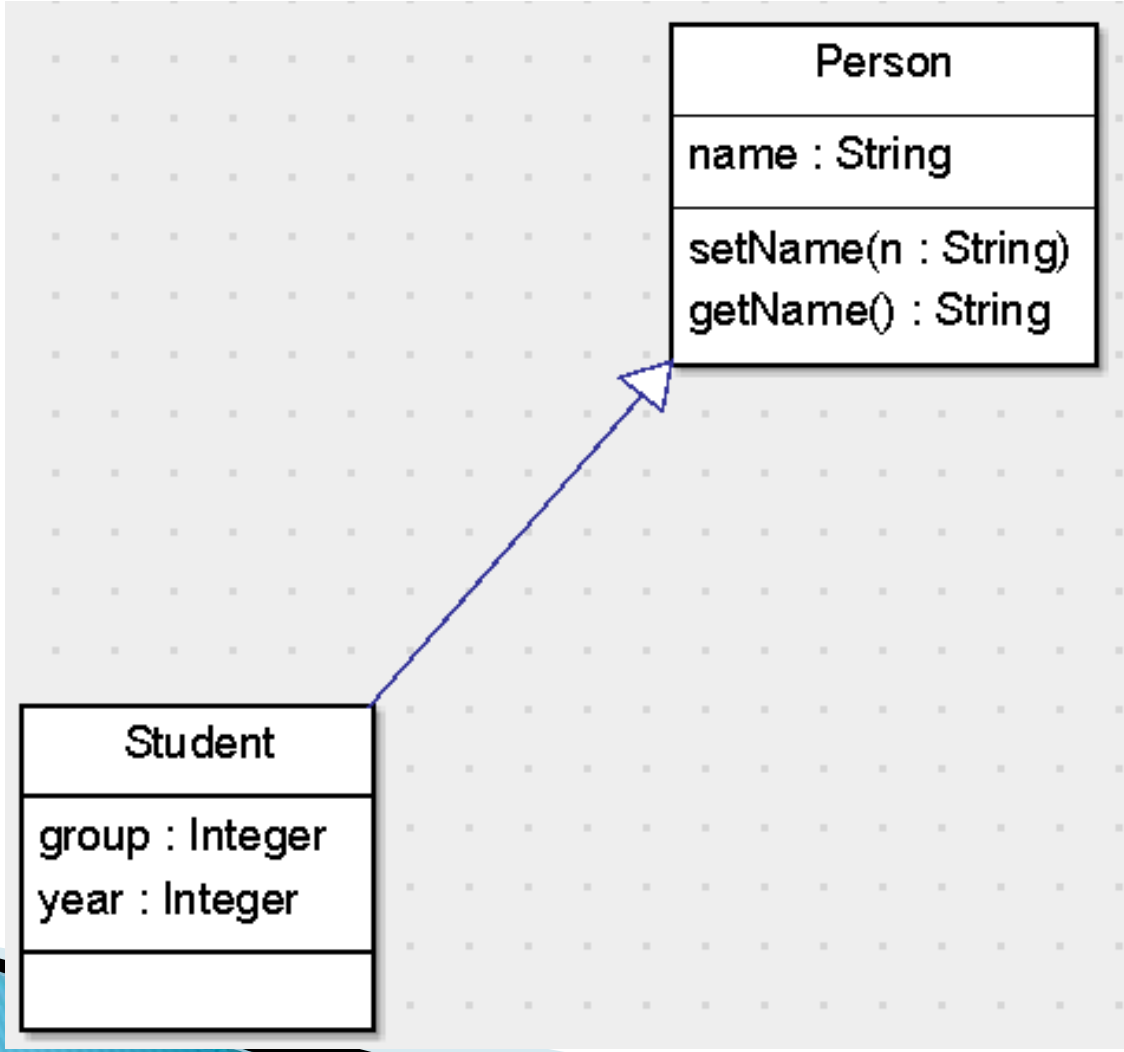
Attributes (private)

Methods (private, protected)

# Relations – Generalization – C#

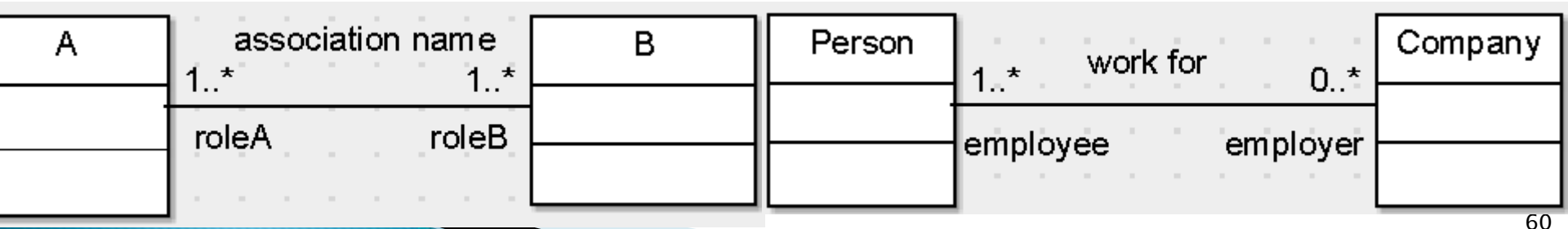
- Modeled concept of inheritance between classes
- Also called the relationship of type "is a"

# ArgoUML – Generalization



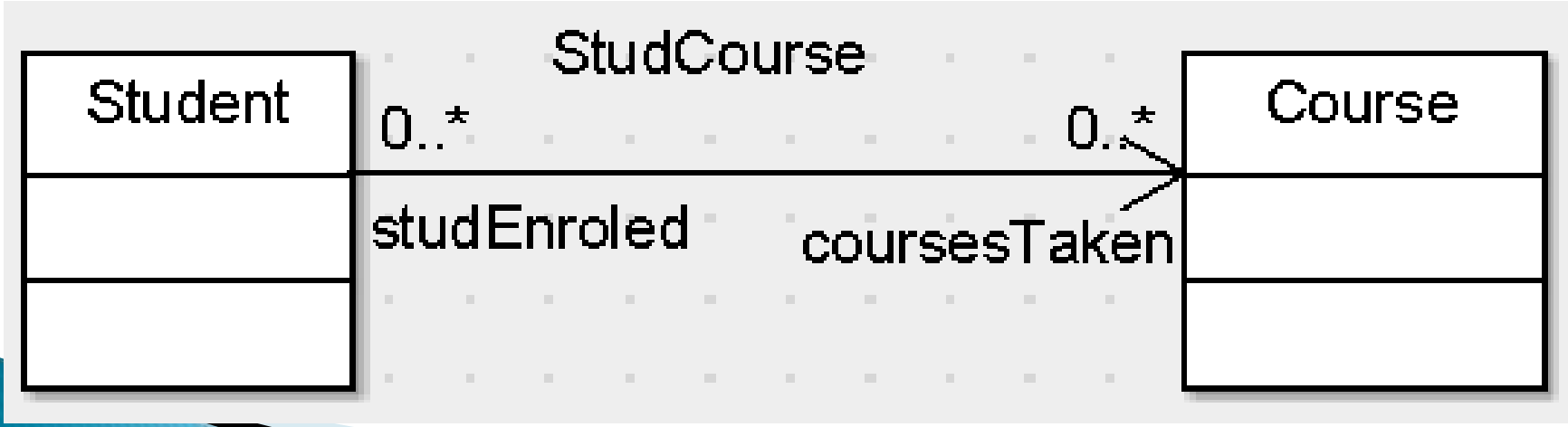
# Association

- Expresses a semantic connection or interaction between objects belonging to different classes
- As the system evolves new connections between objects can be created, or existing connections can be destroyed
- An association interact with objects through its association heads
- Elements:
  - Name: describe the relation
  - Heads of association
    - Name = role of the object in relation
    - Multiplicity = how many instances of a class correspond to a single instance of the other class



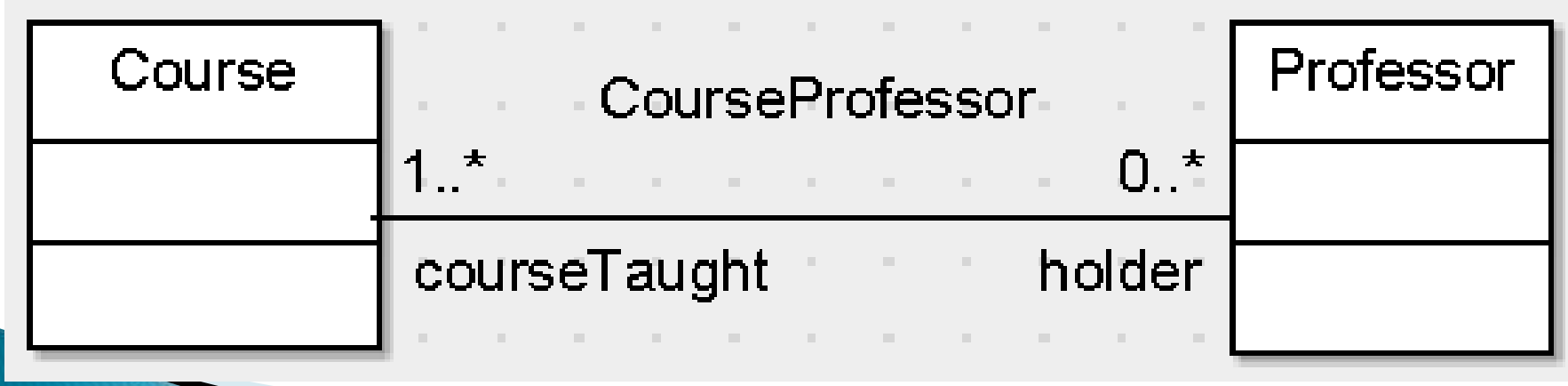
# Association 1

- The relation Student – Course
  - **Student**: follows 0 or more courses, courses know that you follow;
  - **Course**: it can be followed by several students, it does not know students who follow it



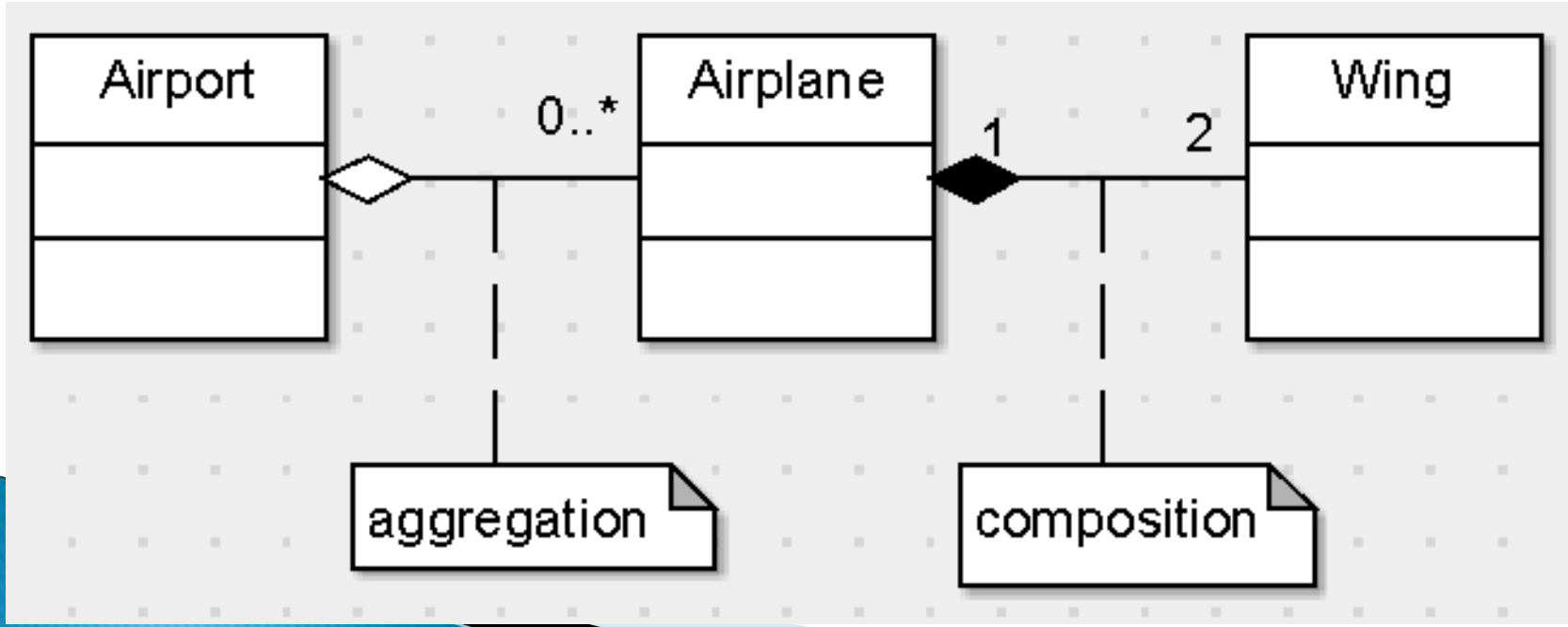
# Association 2

- Relation Course – Professor
  - **Course**: I am taught by a teacher, I know the holder
  - **Profesor**: I can teach many courses, I know the courses they teach

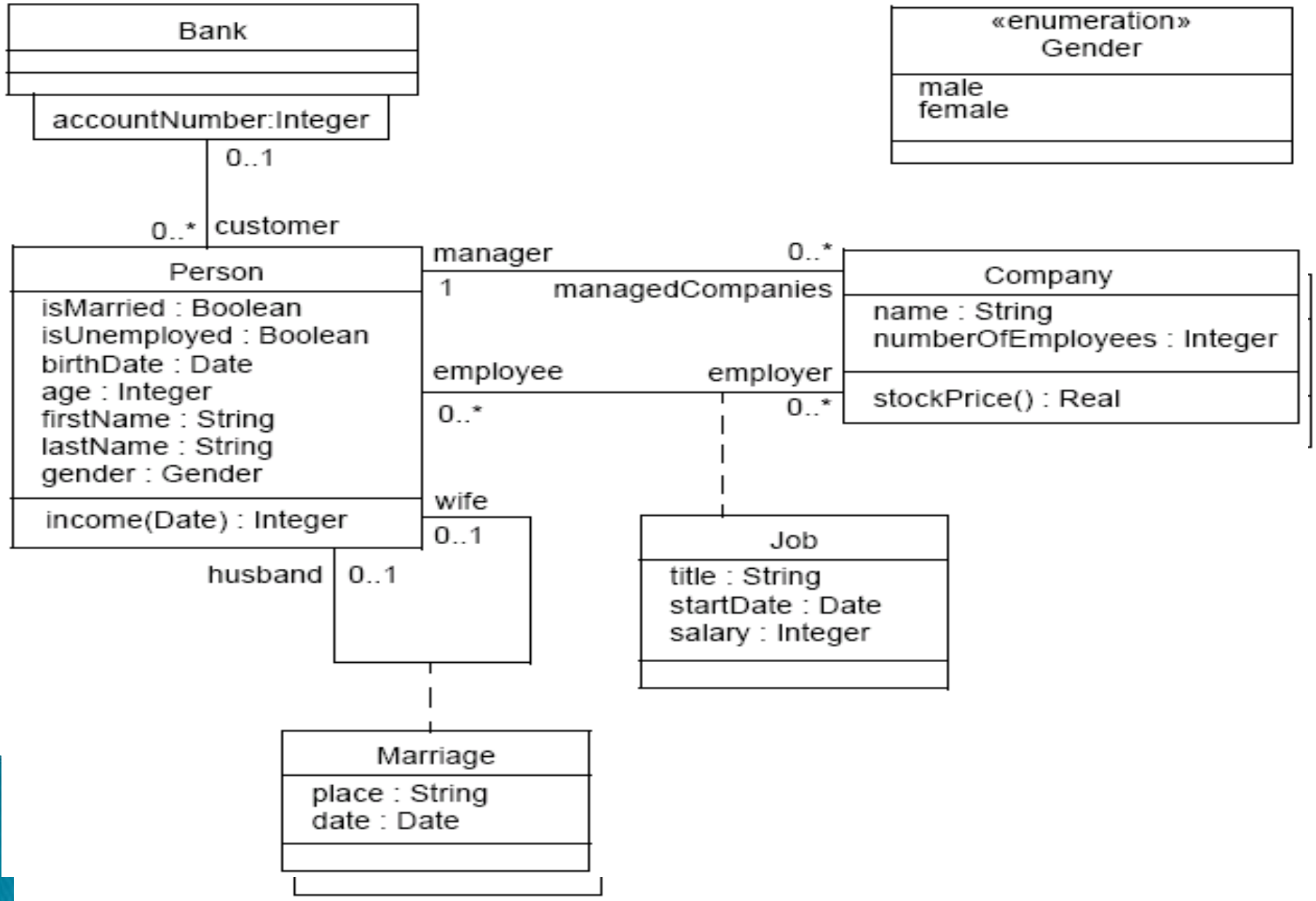


# Aggregation

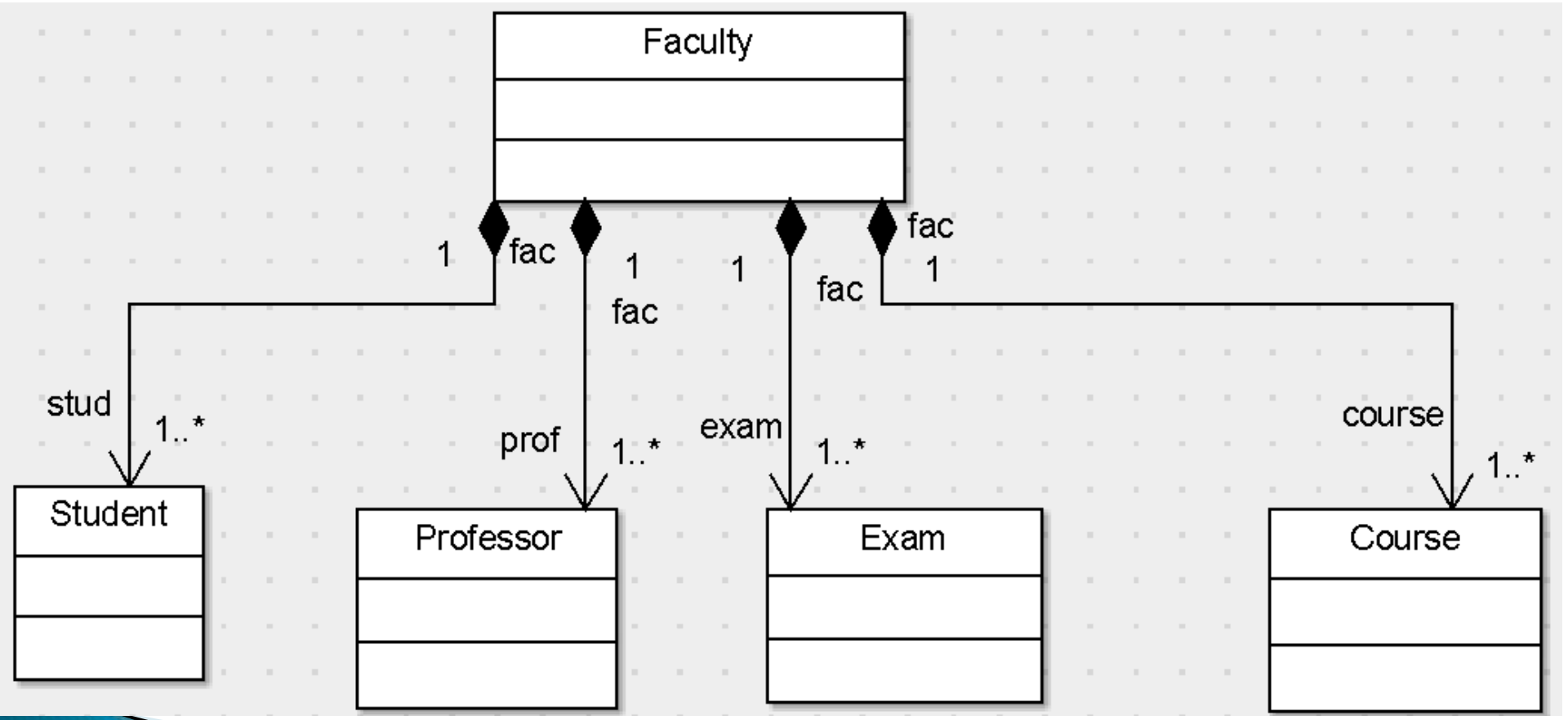
- It is a particular case of the association relation
- It models a part-whole relationship type
- It can have all the elements of partnership agreements, but in general it only specifies the multiplicity
- It is used to model situations in which an object is made up of several components



# Example


Bank
accountNumber:Integer
0..1
0..*
customer
«enumeration»
Gender
male
female
Person
isMarried : Boolean
isUnemployed : Boolean
birthDate : Date
age : Integer
firstName : String
lastName : String
gender : Gender
income(Date) : Integer
manager
1
managedCompanies
0..*
Company
name : String
numberOfEmployees : Integer
stockPrice() : Real
employee
0..*
employer
0..*
Job
title : String
startDate : Date
salary : Integer
wife
0..1
husband
0..1
Marriage
place : String
date : Date

# Composition Relation ("hasA")



# Case Study

- Get students with scholarships
  - Actors
  - Use Cases

# ArgoUML

- Link: http://argouml-downloads.tigris.org/argouml-0.34/
- Zip archive should only be unzipped
- You need to have Java installed
  - In Path you should also have c:\Program Files\Java\jdk1.6.0_03\bin
  - Variabile: JAVA_HOME=c:\Program Files\Java\jdk1.6.0_03\

# Conclusions

- Modeling – Why?
- Graphic Languages
- UML
  - Structural: classes
  - Behavioral: use-case
  - Interactions

# Bibliography


- OMG Unified Modeling Language**TM (OMG UML), Infrastructure,** Version 2.2, May 2008, http://www.omg.org/docs/ptc/08-05-04.pdf
- **ArgoUML User Manual,** A tutorial and reference description, http://argouml-stats.tigris.org/documentation/printablehtml/manual/argomanual.html
- Ovidiu Gheorghieş, Curs IP, Courses 3, 4
- UML Diagrams, Regie.ro

# Links

- OOSE: http://cs-exhibitions.uni-klu.ac.at/index.php?id=448
- ArgoUML: http://argouml-stats.tigris.org/nonav/documentation/manual-0.22/
- Wikipedia